Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

1986-09

MultiSimPC: a multilevel logic simulator for microcompute

Kelly, John S.

http://hdl.handle.net/10945/21990

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

POSTGRADUATE SCHOOL
EY, CALIFORNIA 93943-5002

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MULTISIMPC: A MULTILEVEL LOGIC SIMULATOR FOR MICROCOMPUTERS

by

J. Scott Kelly

September 1986

Thesis Advisor: Harriett Rigas

Approved for public release; distribution is unlimited

T234902

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)
MULTISIMPC: A MULTILEVEL LOGIC SIMULATOR FOR MICROCOMPUTERS

12. PERSONAL AUTHOR(S)
Kelly, John S.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM _____ TO _____		197

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Computer-aided logic design, hierarchical design, event-driven simulation, multilevel simulation

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis describes extensions to a multilevel VLSI logic simulator named MultiSim. Originally developed by Dr. Ausif Mahmood of the Washington State University for large minicomputers such as the VAX-11 780, MultiSim is now operational on desktop microcomputers costing only a few thousand dollars. In addition, MultiSim has been expanded to include provisions for adding user-defined primitive cells to the circuit library, true multilevel circuit expansion, and multiple variations of library primitives within a single circuit.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Harriett B. Rigas	22b. TELEPHONE (Include Area Code) (408) 646-2082 22c. OFFICE SYMBOL 62 RR

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Approved for public release; distribution is unlimited.

MultiSimPC: A Multilevel Logic Simulator for Microcomputers

by

J. Scott Kelly
Lieutenant, United States Navy
B.E.E., Georgia Institute of Technology, 1979

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN
ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1986

ABSTRACT

This thesis describes extensions to a multilevel VLSI logic simulator named MultiSim. Originally developed by Dr. Ausif Mahmood of the Washington State University for large minicomputers such as the VAX-11/780, MultiSim is now operational on desktop microcomputers costing only a few thousand dollars. In addition, MultiSim has been expanded to include provisions for adding user-defined primitive cells to the circuit library, true multilevel circuit expansion, and multiple variations of library primitives within a single circuit.

TABLE OF CONTENTS

I. INTRODUCTION

A. DISCUSSION

Sophisticated computer-aided-design tools have become a virtual necessity for today's logic designer. Current circuit densities are on the order of two hundred thousand transistors on a single chip, as in the Intel 80386. To commit such a design to production without exhaustive simulation before the first chip is laid down in silicon would prove disastrous. In turn, the algorithms for simulation must be able to accurately model extremely complex elements without requiring inordinate amounts of time.

The twin factors of circuit density and development cost have spurred vigorous research in developing tools for documentation, digital design, and simulation techniques. Some of these rely on hardware for rapid calculations while others exploit software techniques to achieve high speed. The emergence of graphics-capable machines has had an enormous impact as well; it is now possible to "draw" a circuit on a computer screen and to have the computer produce photo-ready artwork on a graphical plotter. These graphics-capable machines can also interact with simulators to display timing waveforms and plot the behavior of the circuit for varying inputs.

The research into CAD techniques is beginning to produce some significant products; these include the MAGIC tools and the CRYSTAL timing analyzer from UC Berkeley; the CLL (chip layout language), SIM and ESIM circuit simulators from Stanford; MacPITTS from MIT; and an increasing number of commercial products as well.

Of tremendous import is the ongoing revolution in microprocessors--it is now possible to obtain microcomputers with performance approaching or even equalling the DEC VAX-11/780, and for a tiny fraction of its $200,000 price. Desktop UNIX systems are now commonplace, as are systems with beyond a megabyte of main storage and twenty to thirty megabytes of fixed disk storage. Best of all, in quite a few cases such systems may be had for much less than $10,000.

The advantages of porting high-powered CAD tools onto the new generation of "supermicros" become clear--low cost, high performance and genuinely increased throughput as each designer can possess a dedicated workstation whose processor is not serving several other users. As a result, the competition in this market is becoming quite fierce; in recent years designers working on CAD tools, and in particular those moving them to micros, have played their cards quite close to their chests. There has been relatively little published on simulation algorithms or

speed improvement techniques lest the competition gain an advantage in a new product.

Of those authors who have published algorithms, the event-driven simulation has become the method of choice [Ref. 1, p. 668 and Ref. 2, pp. 182-186]. The event-driven techique exploits "temporal sparseness" [Ref. 1, p. 672] of typical VLSI circuits; that is, in any given circuit, only a few elements will be active at a given time and must be simulated. The rest may be safely ignored with no loss in accuracy.

One particularly interesting program using this technique is called SAMSON2. This program, written for the VAX-11/780, reads a textual description of a user's circuit [Ref. 1, p. 679], then generates C-language routines describing the behavior of the individual circuit elements. These routines are then compiled and linked to the SAM2 simulator to perform the simulation. Using event-directed techniques, the author claims speed improvements of from four to over ten times the speed of SPICE2, depending on the characteristics of the circuit under test [Ref. 1, p. 682].

One of the concerns in implementing a logic simulator is the type of circuit model employed [Ref. 3, p. 76]. The function of the model is to describe the behavior of a circuit component to an input stimulus. The two major contenders are hardware and software models--in the software model, a program segment describes the operation of the

circuit element. The software model will either be at the gate-level, describing the element in terms of its complete gate interconnection, or the behavioral level, where the logical relation of inputs to outputs along with timing data is described [Ref. 3]. The hardware model, by contrast, is intended to be used with a sample circuit to test its behavior in actual operation [Ref. 3].

Of the two, the software model is much less expensive but potentially much slower. This, then, is the tradeoff--the time required for simulating circuits in software against the cost of actually building the component.

However, as simulation techniques improve, interest has dramatically increased in software modeling. Several logic simulators are now available that mix the gate-level circuit representation with behavioral models, offering more attractive speeds of operation. Unfortunately, many of these software programs are still intended for operation on large, expensive computers [Ref. 3, p. 79], so the steady improvement in the capabilities of microcomputers and workstations is being eagerly followed. Several such machines, as well as some of the new generation of microcomputer-based logic simulation programs, will now be considered.

B. COMMERCIAL PRODUCTS

1. EE Designer

EE Designer is Visionics' entry into the microcomputer-based CAD market; at $995, it is one of the

least expensive available. It is a complete package including a schematic drawing and capture section, a netlist generator and a logic simulator. EE Designer is configured to run on an IBM PC (or 100% compatible) or IBM PC/AT, and in order to be genuinely useful a 20 megabyte hard disk is required. The program is completely menu-driven; when initially loaded it presents a menu offering the:

* schematic drawing routine
* schematic capture routine
* penplotter drawing routine
* netlist generator
* simulator
* setup options

The schematic drawing routine takes advantage of the Enhanced Graphics Adapter in the PC- and AT-class machines and produces quite good graphics. The screen area is limited by the relatively small displays available to desktop PC's (a thirteen-inch diagonal screen) but the resolution within that area is acceptable. The drawing area makes up the major portion of the screen with a keyboard command line at the bottom and a menu bar along the right side. Menu-bar items and commands are selected by clicking a mouse-controlled pointer on the desired item. Zooming is available in three fixed ratios (2:1, 4:1, and 8:1); other ratios and mouse-selected viewing areas are not supported.

The display is in color so readability is quite good. With a color plotter, of which several are supported, EE Designer can produce photo-ready artwork.

There is a large repertoire of standard TTL parts as well as several basic CMOS gates available to the user; in addition, EE Designer allows custom circuits to be added to its library. Its simulator is adequate but not particularly quick. It operates considerably slower than the Valid-enhanced PC/AT to be discussed below, but to offset this it does not require the purchase of an expensive coprocessor board.

This is not a particularly easy package to use. Although menu-driven, the user interface can be confusing since it is often unclear (as in many menu-driven systems) as to how to move from one section of the program to another. Some commands require the user to return to the main menu, others do not. While in drawing mode, many of the more important commands must be entered from the keyboard rather than selected from the menu bar; however, to enter a keyboard command requires that the keyboard first be enabled (this is done by selecting "KEYBOARD" from the menu bar with the mouse, then pressing RETURN on the keyboard).

There are three manuals covering all of the various commands but almost no examples showing their use; for a program as complex as EE Designer this is an unfortunate omission. To summarize, EE Designer can be a highly useful

program, but the prospective user should be forewarned that the learning curve will be steep. The reward is a powerful tool at probably the lowest price available for a commercial-grade product.

2. DASH

Another contender in the PC/AT-based CAD workstation market is FutureNet's DASH program. DASH offers similar features to EE Designer, but has the look of a much more highly-polished package. It ought to--in order to do all that EE Designer can (schematic draw/capture, netlist, penplotting and simulation) will require an investment of $17,330. This is, of course, after the customer has already acquired a $6000 IBM PC/AT system with a 20-megabyte hard disk, Enhanced Graphics Adapter and long-persistence color monitor.

Once all the money has been spent, the user will have a very sophisticated logic design tool available. The user interface is very smooth, combining a menu bar/mouse interface with the conventional keyboard commands. DASH slips easily from module to module, and although menus are employed they operate in a much more sophisticated fashion than EE Designer. Invoking a menu causes a window to be placed on the screen with the available choices; picking one of the selections will place another window on the screen with the subchoices for the selection just made. Thus, the user is not only aware of which command is active, but where

he or she stands in relation to the other commands. The effect is not unlike the windowing user interface of the Macintosh and related machines--the feel of the program is quite intuitive and the learning curve is consequently much less steep.

Unfortunately, in the area of simulator performance it is obvious that the host machine is a 6 megahertz IBM PC-AT. EE Designer should be able to simulate circuits just as quickly as DASH for much less money; however, neither is overly fast. In short, DASH is an excellent tool but it is difficult to justify the steep entry price.

3. Valid PC/AT

The reason it is difficult to justify the entry price for DASH is this machine. For about two-thirds the price, Valid Logic Systems places a coprocessor board into the IBM PC/AT (or 100% compatible) with additional memory, installs the UNIX operating system and its GED (Graphics EDitor) and simulator software and produces a significantly faster machine. The coprocessor board offers the National NS32032 microprocessor which is several times faster than the Intel 80286 of the host PC/AT; the result is a significantly faster simulation. Operation of this machine is identical to the Valid SCALDstar workstation discussed below; the major difference is the host machine and the display (the IBM uses a 13-inch, 640 x 400 pixel display).

The major difference between the SCALDstar and the Valid PC/AT is the appearance of the simulation results--

while the SCALDstar (and the above two packages) produce a detailed graphical plot of the output waveforms, the Valid PC/AT uses a simple ASCII-character display. For instance, a square wave would appear as:

<variable name>: _____/^^^^^^^_____/^^^^^^^

This is less than impressive considering the caliber of the GED displays and the capabilities of the hardware employed. It seems doubly unfortunate when one considers that an accurate plot of the simulation results are, in the end, much more important than a tidy circuit drawing since if the circuit fails to operate correctly, the design effort was largely wasted.

4. Valid SCALDstar

The Structured Computer-Aided Logic Design (SCALD) concept was developed at the Lawrence Livermore National Laboratories "specifically to address the new needs of the logic designer. It consists of both a design method and a set of hardware and software tools to support that method" [Ref. 4, p. 135]. The hardware tool is the MC68000-based SCALDstar workstation from Valid Logic Systems. This is one of the first microprocessor-based CAD workstations and represented a significant departure from the previous norm--graphic terminals attached to a mainframe or large minicomputer (such as a VAX-11).

The SCALDstar is based on the Intel MultiBus; the MC68000 operates as a file server and cluster controller for

the Intel 8086 processors with which the graphics displays and simulations are performed. The ECE Department's two SCALDstars are fairly representative installations in that each possesses four megabytes of main storage, seventy megabytes of fixed disk storage and a magnetic tape cartridge for archival use. The Department's machines are further equipped with Ethernet links so that drawings and other data may be exchanged between them. The SCALDstar employs a 19-inch monochrome screen for circuit design work (a second 19-inch color monitor is used in custom design of the actual silicon with a different set of software tools). This monitor supports 1024 x 1024 pixel resolution, equivalent to almost twice the horizontal and nearly three times the vertical resolution of conventional microcomputer displays. Consequently, the level of visual detail in the graphic editor is most impressive.

The software tools provided are highly-developed (the Graphic EDitor, or GED, is in Version 8.0) and highly capable. GED's display area is immense, with a row of mouse-selected menu options along the right side and a command line along the bottom. Display modes are endlessly flexible; the user can flip, rotate, invert and show alternate versions of circuit elements. Also, GED is surprisingly easy to use for a program of its complexity. The mouse-based interface is fairly easy to learn and there are few routinely-used commands in GED that are not

available as menu selections (as opposed to EE Designer, where few of the commands are available as menu items).

Preparing a completed circuit for simulation can be done in two ways:

* by selecting the SIMULATE command from GED; this causes GED to invoke the circuit compiler and set up the simulator screen.
* by exiting GED, performing the circuit compilation manually and entering the simulator directly.

The tradeoff between the two methods is speed (and to a lesser extent, display capability). SCALDstar is a multitasking machine based on the UNIX operating system; if the simulator is called from GED, GED remains active while a new process, corresponding first to the compiler and then the simulator, is launched. The advantage of this is that signals the user wishes to observe may be "picked up" with the mouse and "dropped" into the simulator window; as noted, this makes for a very intuitive, easy to use tool. However, he or she pays dearly for this convenience in terms of execution speed. Further, with most of the window taken up by GED, only a few lines may be observed on the simulator at a time. If the user wishes to forsake a bit of convenience, invoking the circuit compiler and simulator directly will yield a dramatic increase in execution speed. This is particularly significant for large circuits where compilation times of fifteen to twenty minutes (or in some cases, much, much longer) might otherwise result. As an

added benefit, the entire screen is now available to observe circuit lines; as many as forty separate lines may be displayed at once.

The simulator operates fairly quickly but it will not operate as fast as the PC/AT machines; while the AT's, even without a coprocessor, are equipped with 80286 processors, the SCALDstar is operating an 8086. Even in the best of circumstances, the older 8086 will simply not be able to keep pace with the newer microprocessor.

Finally, the item having the greatest impact on the SCALDstar at the present time is its $50,000 price. Even with the caliber of software provided, which is frankly exceptional, newer, faster and cheaper machines are arriving.

5. ISI Graphic Workstation

In its class, where the benchmark is the Sun graphics workstation, this is perhaps the most impressive new machine on the market. From Integrated Solutions, this $30,000 workstation is based around the Motorola MC68020 32-bit microprocessor and is equipped with the MC68881 floating point processor. The Department has acquired three of them, two with 70 megabyte fixed disks and the other, which is employed as the master and file server, is fitted with 120 megabytes of fixed disk. Each machine has four megabytes of main storage, an Ethernet link, an interface to a high-resolution color graphics plotter and other minor equipment.

Like the SCALDstar, its display resolution is a full 1024 x 1024 pixels but in color as well. The ISI Workstation runs BSD 4.2 Unix as its native operating system with both the C and Bourne shells provided. In addition, it offers a desktop-type iconic user interface that can support up to eighteen concurrently operating tasks, each in its own window. This machine offers a substantial percentage of the VAX-11/780's computational power and offers tremendously exciting capabilities for the designer (for a price, of course).

The Department presently has the Berkeley MAGIC VLSI design tools implemented on the ISI's. Although MAGIC is intended for custom design at the level of silicon, these machines are more than capable of supporting a gate-level design package. It is significant to note that MAGIC was developed for, and intended to run on, the VAX-11 family. The fact that these tools are operating on the ISI workstations with little loss in performance is an indication of just how much capability has been designed into the ISI machines.

II. THE MULTISIM PACKAGE

A. BACKGROUND

Unfortunately, two common threads link all of the previously-discussed workstations and software packages: they are either prohibitively expensive, irritatingly slow or both. Schools, particularly, are hurt by the inability to afford more than a handful of Sun-caliber machines for laboratory use. Further, the lackluster performance of existing simulation routines make these already crowded machines even busier as the students must schedule long sessions on slow machines. (This is precisely the situation faced by the Department over the two SCALDstar workstations. To help improve matters, the Department was able to procure only an additional three ISI workstations.)

Distilling the above discussion, a clear requirement emerges--a fast simulation algorithm that can run on inexpensive, easily-obtained hardware.

The explosion that brought the Sun, the ISI and IRIS to the marketplace has not neglected the lower end of the spectrum, either; it is arguable that the most dramatic progress has been made in the so-called "personal" computers. Sixteen bit processors, fixed disk drives and acceptable graphics are now the rule rather than the exception in such machines. Best yet, the current crop of

personal computers carry extremely attractive price tags; for the price of one SCALDstar, it is possible to buy twelve IBM PC-AT machines, each with a megabyte of main storage and twenty megabytes of fixed disk storage. Clearly, the hardware portion of the requirement has been met.

B. THE TOOLS

During the period from 1981 through 1985, as his doctoral research at the Washington State University, Ausif Mahmood constructed a VLSI logic simulator that had been developed by Dr. H. B. Rigas. This simulator is based on **Ulrich's Algorithm** [Ref. 5], which exploits the fact that in a typical circuit, only about 2.5% of the various elements that compose the circuit will change state at any instant. Thus, only those elements whose states are actually changing need to be modeled and the rest may be ignored. The resultant "event-driven" simulation operates approximately forty times faster than an exhaustive element-by-element ("compiler-driven") simulator with no loss in accuracy.

Further, the simulator is capable of multilevel operation. Circuit elements are constructed hierarchically, building complex modules out of simpler ones, and this process may be carried to an arbitrary number of levels. However, when simulating such a circuit, the user is faced with an important question--how detailed does the simulation need to be? Is the high-level description adequate? If not, the circuit description language allows the user to

expand a module to any level of detail, even to the most basic primitives that compose each submodule.

There are three components to the simulation system, called MultiSim:

* The VLSI-Oriented Hardware Language, or VOHL. This is the syntax used to describe a circuit to the simulator;

* The circuit compiler. This is a separate program from the simulator, and translates the VOHL circuit description into the data structures used by the simulation program; and

* The Timing Wheel simulator. This is the event-directed simulation program.

Both the compiler and simulator are written in the C programming language, and the original target machine was the VAX 11/780 minicomputer running the BSD 4.2 UNIX operating system.

1. VOHL Syntax

The VOHL description language is a user's tool to describe digital circuits to the compiler (a complete discussion of VOHL is contained in Mahmood [Ref. 6, pp. 161-170] with a reference card in Appendix A). Figure 1 is a typical VOHL circuit, in this case describing a JK flip-flop. The MODULE keyword identifies the beginning of a new module, with the INPUTS and OUTPUTS to the flip-flop listed following the appropriate keywords.

The TYPES keyword is used to declare variations on system primitives. For example, several different NANDs may be declared with their parameters subsequently set by the

```
MODULE: JKFF;
INPUTS: J, K, CLK;
OUTPUTS: Q, Q';
TYPES: INTERNALS: S1, R1;
   {
   S1 = NANDTHR(Q', J, CLK);
   R1 = NANDTHR(CLK, K, Q);
   Q, Q' = RSCELL(S1, R1);
   }
DEFINE: ;
INITIALIZE: S1=0; R1=0;
PRINTOUT: J, K, CLK, Q, Q'

MODULE: RSCELL;
INPUTS: S, R;
OUTPUTS: Q, Q';
TYPES: ;
   {
   Q = NAND(S, Q');
   Q" = NAND(Q, R);
   }
END;
```

Figure 1. A J-K Flip-Flop in VOHL

```
MODULE: JKFF;
INPUTS: J, K, CLK;
OUTPUTS: Q, Q';
TYPES: INTERNALS: S1, R1;
   {
   S1 = NANDTHR(Q', J, CLK);
   R1 = NANDTHR(CLK, K, Q);
   Q, Q' = RSCELL(S1, R1);
   }
DEFINE: NANDTHR: RISEDELAY(0,0)=3;
EXPAND: RSCELL: NAND: RISEDELAY(0,0)=1,
                      FALLDELAY(0,0)=2, FANOUT=15;
INITIALIZE: S1=0; R1=0;
PRINTOUT: J, K, CLK, Q, Q'
END;
```

Figure 2. The J-K Flip-Flop
Using Expansions

DEFINE keyword (discussed below). The INTERNALS keyword is used to declare "scratchpad" variables (those which are neither inputs or outputs).

Note that in the circuit description, each statement is of the form:

output variable = primitive name (input variables) ;

where the primitive name corresponds to a logical function. The compiler is not equipped to evaluate subexpressions within the input variable list, as this could quickly become a hopeless task; therefore, the internal variables act as "placeholders" for use in later expressions. For example, in the JK flip-flop, since the expression

Q, Q' = RSCELL(NANDTHR(Q',J,CLK), NANDTHR(Q,K,CLK))

is too complicated, it is broken up into the form shown.

The circuit body is preceded by the opening brace and followed by the closing brace.

After the circuit body of the main module come the simulation control specifications. It is possible to DEFINE parameters of primitives, such as rise or fall delay, for more detailed analysis or for fault investigation. Primitives may also be EXPANDed to any lower level, all the way to AND-OR-INVERT gates if desired. Parameters may also be specified for the EXPANDed primitives if the user wishes. Figure 2 illustrates how these particular keywords are used; in this example the RSCELL is assumed to be a library primitive, so its description is not included after the J-K flip-flop.

The INITIALIZE keyword is used to set up initial conditions for the circuit. Both internal and output variables may have values preassigned in this manner. Lastly, PRINTOUT specifies the variables to be displayed during simulation.

Note that submodules may be included in the VOHL description--just as a large program is constructed of less complex procedures (which may themselves be made of sub-procedures, and so on), a complex circuit may be built up in precisely the same fashion. The submodules are included in order of precedence, with the higher-level circuits preceding the lower.

2. Compiler

After creating a VOHL circuit as an ASCII text file, the user invokes the compiler. The compiler performs three tasks in the following sequence:

* checks for requests to expand a primitive to greater detail;
* performs any such expansions as macro substitutions of low-level VOHL code, and performs macro substitutions of user-defined submodules;
* parses the resulting single-level VOHL circuit and builds the data structures used by the simulator (discussed below).

Expansion requests come in two forms: explicitly, by the user issuing an EXPAND keyword; or implicitly, by the inclusion of submodules in the VOHL circuit. In the first case, the compiler scans the VOHL circuit for EXPAND

keywords; if any are found, the module names in each EXPAND line are placed in an "expand table." The compiler checks the second case by scanning for additional MODULE keywords corresponding to user-defined submodules; any that it finds are also added to the expand table. In addition, the compiler moves the structural descriptions of these submodules into a temporary library for use as described below. It is significant to note that an EXPAND keyword will only carry out expansions to the next lower level; if further expansion is required additional EXPANDs must be issued.

As an example, Figure 2 shows a JK flip-flop that contains an RS flip-flop as one of its elements. Here the user is interested in examining the behavior of the JK while paying particular attention to the RS, hence the expansion request for the RS flip-flop. Note that particular values for the components making up the RS have been called for as well. When the compiler encounters this circuit, it will trap the EXPAND keyword and place RSCELL in the expand table.

Observe that at this point, although the first steps have been taken in the expansion process, the user's VOHL circuit is still in hierarchical form. The next step is to expand the hierarchical representation into a single level; this is done by using the lower-level modules marked in the expand table as macros. For system primitives, the lower

level modules will be found in the auxiliary file STRUC, and for user-provided submodules the descriptions will be in the compiler-generated temporary library. The compiler enters the appropriate library to extract the submodule definition, then substitutes the submodule code for the original circuit line.

Again considering the JK flip-flop, since the RSCELL is in the expand table the compiler substitutes its VOHL structural code for the expression

```
Q, Q' = RSCELL(S1, R1);
```

The RSCELL's parameter list in the circuit expression is matched with the structural code's INPUTS and the output variables are matched with the structural code's OUTPUTS.

```
MODULE: JKFF;
INPUTS: J, K, CLK;
OUTPUTS: Q, Q';
TYPES: INTERNALS: S1, R1;
   {
   S1 = NANDTHR(Q', J, CLK);
   R1 = NANDTHR(CLK, K, Q);
   Q = NAND(S1, Q');
   Q' = NAND(Q, R1);
   }
DEFINE: NANDTHR: RISEDELAY(0,0)=3;
        NAND: RISEDELAY(0,0)=1,
              FALLDELAY(0,0)=2, FANOUT=15;
INITIALIZE: S1=0; R1=0;
PRINTOUT: J, K, CLK, Q, Q'
END;
```

Figure 3. The Expanded JK Flip-Flop Generated from Circuit of Figure 2

Performing the substitution produces the circuit shown in Figure 3; note that the RSCELL has been replaced by a pair of NANDs. Also, since the RSCELL no longer appears, the simulation control specifications that applied to the NANDs in the EXPAND line have been moved to the DEFINE line instead.

In order to better understand the actual compilation phase, it is first necessary to introduce the concept of the **descriptor record.** These records describe how the components that make up the circuit are interconnected as well as the behavior of each component. Figure 4 represents a single record; the various fields include the type of primitive function the record describes (an AND gate, for instance), pointers for up to two inputs and fields for their values, such parameters as rise and fall delay, technology type, fanout and so on. The **header** pointer is used to tie the descriptor records together. Not shown in Figure 4 is a field for an extension pointer. A single record can handle only two inputs and one output, but a primitive function may have an arbitrary number of inputs or outputs. To expand the number of inputs and outputs, several records are used to describe the primitive. For example, a three input NAND gate requires two records--the first two inputs are handled in the first descriptor and the third is placed in the second descriptor (the extra input field and output field are not used).

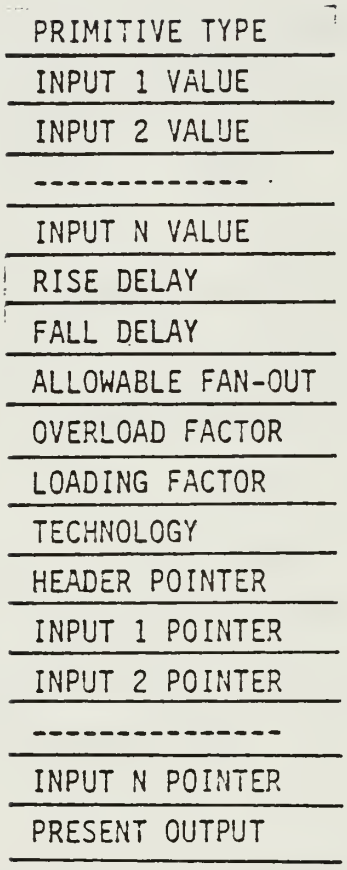


Figure 4. The Descriptor Record
[Ref. 6, p. 42]

There is also a degenerate case of the descriptor record; this is the "read-input" function. It uses only one of the input fields and performs no logical function--the input is simply passed as an output to the next descriptor in the list. This descriptor, clearly, is used to represent inputs to the circuit.

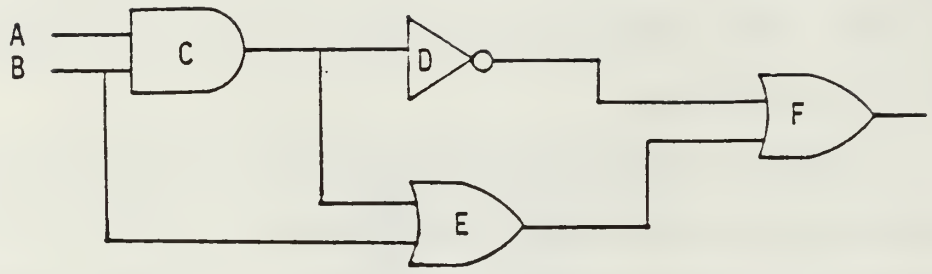
A simpler circuit than the JK flip-flop will be used to demonstrate the compilation phase; this is the four-gate combinational circuit shown in Figure 5 in both schematic and VOHL form. Figure 6 demonstrates how the descriptors are linked together. From start to finish, the compiler assembles the descriptor interconnections in the following manner:

Compiler reads X1 = AND(A, B);

1. A "read-input" descriptor is built for input A, then another is built for input B. Each header pointer is initialized to point back to the respective descriptor.
2. A Type-1 (AND) descriptor is built for the AND gate C. Since C has only two inputs and one output, no additional descriptor is needed and the extension pointer is left as NULL.

3. Since A is an input to the AND gate C, A's header pointer gets the address of C. Further, since A is used nowhere else, C's Input-#1 pointer is pointed back at A.

4. Likewise, B's header pointer gets the address of C. At this point, B is not used anywhere else either, so C's Input-#2 pointer is pointed back to B.




```
MODULE: COMPILATION_DEMO;
INPUTS: A, B;
OUTPUTS: Q;
TYPES: INTERNALS: X1, X2, X3;
   {
   X1 = AND(A, B);
   X3 = OR(X1, B);
   X2 = INVERT(X1);
   Q = OR(X2, X3);
   }
DEFINE: ;
INITIALIZE: X1=0, X2=0, X3=0;
PRINTOUT: A, B, Q;
END;
```

Figure 5. The Four-Element Demonstration Circuit [Ref. 6, p. 43]

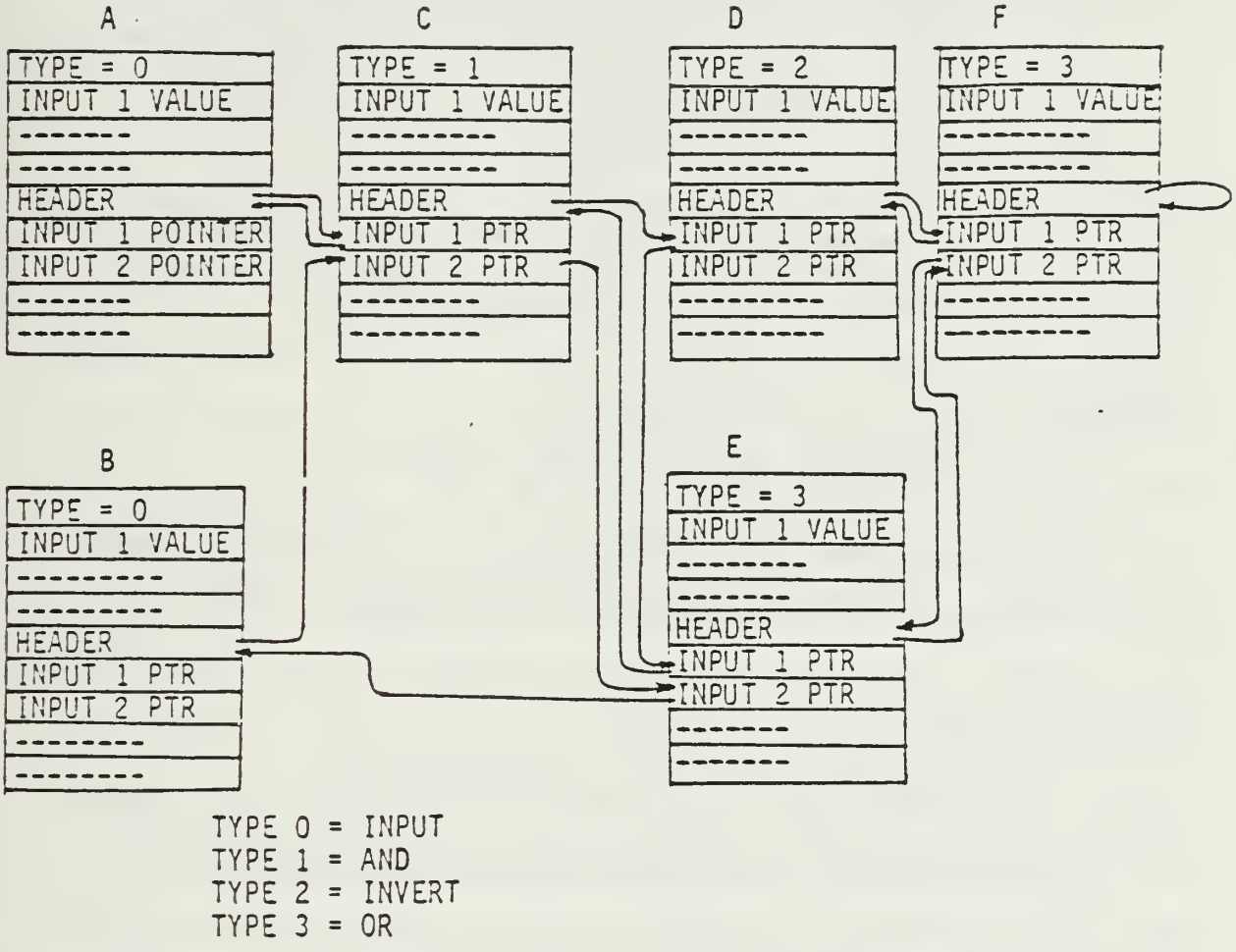


Figure 6. Descriptor Interconnections
for the Demonstration Circuit
[Ref. 6, p. 43]

Compiler reads X3 = OR(X1, B);

1. A descriptor is built for the OR gate E. As with the AND gate, no extension descriptors are necessary.

2. B is an input to E, so the pointer loop B.header --> C, C.input2 --> B is broken to insert the gate E. C's Input-#2 pointer is given the address of E, then E's Input-#2 pointer is given the address of B. Thus, the pointer loop is now B.header --> C, C.input2 --> E, E.input2 --> B.

3. E also takes the output of the gate C as an input, so C's header pointer gets the address of E. As yet, no further connections with E need to be made so E's Input-#1 pointer points back to C.

Compiler reads X2 = INVERT(X1);

1. A descriptor is built for the inverter D.

2. C is also used as an input to D, so the pointer loop C.header --> E, E.input1 --> C is broken to insert the inverter. The new pointer loop is C.header --> D, D.input1 --> E, E.input1 --> C.

Compiler reads Q = OR(X2, X3);

1. A descriptor is built for the OR gate F.

2. Since D is an input to F, D's header gets the address of F. F's input-#1 pointer points back at D.

3. Likewise, E's header gets the address of F, and F's Input-#2 pointer points to E.

Compiler reads simulation control specs.
Compiler reads END; and stops.

Note that the OR gate F's header pointer still points at itself. Therefore, F is not used as an input by anything; it must be a circuit output. In this case it is the only circuit output, so all of the other header pointers

have values other than the descriptor records of which they are a part. Note also that the pointers are arranged in loops. This is a convenient way of identifying the end of a list while traversing--if the start point is reached the traversal is complete.

3. Simulator

Each of the pointer loops built by the compiler corresponds to an "activity stack" [Ref. 6, p. 16] in the simulator. A state change in a circuit element causes the pointer loop (activity stack) originating from that element to be flagged as "potentially active" [Ref. 6] and scheduled for evaluation. The simulator checks all of the activity stacks, in order, for any elements requiring evaluation; after all the stacks have been checked the cycle repeats until all circuit inputs have been exhausted. Because of this cyclic behavior the simulation algorithm has been dubbed the "timing wheel" [Ref. 6, p. 18]. The great power of this algorithm reveals itself in that those stacks not marked for evaluation may be ignored; as noted earlier, most of the time only about 2.5% of the circuit elements in an average circuit are actually changing state. The result is better than an order-of-magnitude speed increase over an exhaustive simulation.

Using the example of Figures 5 and 6, altering the input A will result in the AND gate C being marked as "potentially active" and scheduled for evaluation. However,

if input B is altered then both C and the OR gate E must be scheduled, since both gates are in the pointer loop (activity stack) beginning at B. Thus, the activity stacks originating at both gates are marked "potentially active." Assuming that changing B caused an output change at C, D becomes potentially active, and so on.

If C did not change, however, D will not become potentially active and may be ignored. Likewise, E will not become potentially active unless made so by a change at B. If this is the case, the simulator may quickly skip over them to subsequent activity stacks.

C. THESIS STATEMENT

The compiler and timing wheel simulator form the software nucleus of a high-powered CAD/CAE system. Though originally developed on a small mainframe, the attraction of placing this kind of power into the new breed of sixteen-bit desktop microcomputers is irresistible. Further, since both the compiler and simulator are in their original versions and lack a number of features (most notably the ability to add additional primitive cells to the library), installing such features is of great importance to improve their usefulness.

To these ends, it is the purpose of the research described in this thesis to:

* install the MultiSim system with its VOHL compiler and timing wheel simulator onto an IBM PC-AT desktop microcomputer;

* install library addition routines into the compiler;

* improve the user interface to MultiSim to both ease its use and increase its flexibility; and

* improve the hierarchical capabilities of the MultiSim system.

III. MICROCOMPUTER IMPLEMENTATION

A. PORTING TO A MICROCOMPUTER

The first step in building the microcomputer-based CAD/CAE system was to port the source code for the VOHL compiler, the timing wheel, and the various auxiliary files onto a microcomputer. The IBM PC-AT was selected due to its wide acceptance and reasonable price.

The development environment for the Computer Laboratory's PC-AT systems includes the Microsoft MS-DOS operating system (version 3.1), the Mansfield Software Group KEDIT programming editor (version 3.51), the Lattice C compiler (version 3.10) and the Digital Research GEM (Graphic Environment Manager) system. The communications software used to perform file transfers with the VAX is Simterm; the initial downloads of both the VOHL compiler and simulator were performed with this program. In addition, Hayes Smartcom II is available for communication with the VAX when VT-100 terminal emulation is required.

For hardware, the development machine contains an Intel 80286 with an 80287 FPU, one megabyte of main storage, twenty megabytes of fixed disk, one high-density and one normal double-density floppy disk drive. The machine is also equipped with the Enhanced Graphics Adapter (EGA) and long-persistence monitor.

The original author's choice of C as the implementation language paid rich dividends during the porting process, as it was ultimately necessary to change only two lines of C source code (excluding file pathnames) out of almost two hundred thousand bytes of code to enable the two programs to run successfully. The offending lines dealt with ASCII string input; in BSD 4.2 C, it is possible to read strings and numbers in the same logical line using fscanf--Lattice apparently (though this is not documented) requires that strings and numbers be read with separate fscanf statements. Beyond flattering Dr. Mahmood's coding techniques in using standard structures and statements, the experience was a most impressive display of the portability of C code across major system boundaries.

Though the IBM PC-AT is equipped with a fixed disk, for speed considerations it proved desirable to exploit the virtual-disk capability of the machine. (A virtual-disk is a block of main memory that operates as an extremely-high-speed disk.) The VOHL compiler performs large numbers of disk accesses, and even with the fixed disk execution speed was rather less than desired. Also, it is desirable to maintain a consistent user interface to the MultiSim system; the commands to invoke the compiler and simulator directly require different parameters in different forms, which risks confusing the user (particularly a student in a laboratory setting) and increases the possibility for error.

Accordingly, an MSDOS shell was built around the two programs with the following features:

1. It enables both the compiler and simulator to be invoked by a single command, MODEL. MODEL has two parameters, the name of the circuit to be modeled and the operation to be performed. There are three options:

   E: End-to-end. The shell will invoke the compiler for the named circuit, then cause the simulator to execute. For the simulation, there must be a file of circuit inputs available with the same name as the circuit with a ".IN" extension. The results of the simulation will be piped to the screen, as well as saved in the file <circuit name>.OUT.

   C: Compile only. The shell will invoke the compiler for the named circuit. The results will be saved as <circuit name>.SIM.

   S: Simulate only. The circuit must have been previously compiled so that the .SIM file is available; also, the input file (.IN) must also be present. The simulation results will be in <circuit name>.OUT.

   For example, if the JK flip-flop of Figure 1 was saved in the file JKFF, it could be compiled with the command:

   ```
   MODEL JKFF C
   ```

   to see if there were any errors in the VOHL syntax. If the inputs to the circuit are contained in the file JKFF.IN, the command:

   ```
   MODEL JKFF E
   ```

   will compile the VOHL description, then perform the simulation and return the results in JKFF.OUT.

2. It moves all of the various working files, the VOHL circuit description and the circuit inputs to the virtual-disk. Upon program termination, the shell copies all appropriate files back to the fixed disk for permanent storage.

The compiler was modified to address all files from the virtual-disk, as was the timing wheel. Table 1 indicates the speed gained by utilizing the virtual-disk for a complex circuit; the execution time on the Department's VAX 11/780 is also provided for comparison. As may be observed, using the virtual-disk, execution times quite favorable to the VAX are now possible on a desktop computer costing only a few thousand dollars, heralding a very exciting new dimension in computer-aided design and engineering. No longer is it necessary to spend exhorbitant amounts of money for extremely powerful design tools--such tools may now be had by small companies and schools, perhaps even individuals, on off-the-shelf equipment available at any computer store, and at extremely attractive prices. By way of comparison, the Department's two Valid SCALDstar workstations cost $12,000 per year merely to maintain.

TABLE 1.
THE ALU BENCHMARK
(400 gates, 100 clock cycles)

	IBM PC-AT fixed disk	IBM PC-AT virtual disk	VAX 11/780
compile	3:51	1:49	3:00 (average)
simulate	:05	:05	:05 (average)

All of the above times were obtained with a stopwatch; the VAX execution times varied with the load on the system,

so the average of many passes is shown. The execution times on the single-user, single-tasking IBM PC-AT never varied more than one second from the times shown.

B. ENHANCEMENTS

The more challenging work commenced after the programs were successfully ported to the IBM PC/AT. Though powerful and versatile, the tools still possessed significant limitations. While they were chosen on the basis of their usefulness, there were only fifteen available primitives. There was no simple way to add additional circuits to the library although the original design made provisions for future additions in the design of its various data structures. Finally, all user interaction with the programs were text-only; the compiler accepted a VOHL circuit description in the form of an ASCII text file as input, and the timing wheel returned a table of outputs vs. time--another ASCII file.

In many cases, a user prefers to draw a circuit on the computer screen, selecting existing cells from the primitive library and using the mouse to make cell connections. The compiler could then capture the circuit schematic from the screen and compile the circuit in the normal fashion. The user would also be provided the option to save the new cell as a primitive. The timing wheel would then model the behavior of the circuit, but instead of returning a table of outputs, would display the simulation results on the screen graphically.

The idea of a graphical user interface was certainly fascinating but it quickly proved to be unmanageable. The currently-available graphical programming environment for the IBM PC family is the Digital Research GEM system; unfortunately, this is a highly immature system with virtually no documentation for its hundreds of complex function calls. Further, it possesses significant limitations for use as a foundation for CAD tools:

* While it is possible to move about in a display "window", as if looking over different areas of a circuit layout, there is no capability for zoom-in/zoom-out.

* There is no way to build "hooks" or "handles" onto a bit image such as a NAND gate, where GEM can detect that an input line or output line (as opposed to the entire image) has been selected.

* There is no way to flip, rotate or invert images using GEM function calls.

* Most importantly, though GEM is itself written in C, it requires assembly-language bindings to the various C compilers available for MS-DOS (including Lattice). At present, only the "small-memory-model" bindings are available; these bindings allow only a 64K partition of memory for all code, data, stack and heap space.

The GEM code merely to open a window on the screen uses nearly all of that 64K allotment--the VOHL compiler alone already requires the large memory model (up to one megabyte address space for code and data). The large-model bindings are promised as updates to the GEM distribution but were not available in time for use in this project.

Accordingly, most of the effort was directed at implementing the library addition routines. The addition

routines, too, required a fresh evaluation of the compiler and simulator. In the interest of speed, the simulator maintains two representations for each primitive cell:

* the structural description. The structural description for the primitives is in VOHL code, with which the primitive's behavior may be precisely modeled; and

* the block description. The block description includes C-language procedures to simulate the overall behavior of the primitives, rather than the precise detail of every subelement. Detail of simulation has been exchanged for execution speed. The block description also includes a block delay matrix for each primitive; this is a table of delays from each input of the primitive to each output and is maintained in the auxiliary file BLDEL.

The block-description procedures are maintained in a separate file called BLOCK which is included into the simulator at compile-time. Unfortunately, since the block procedures are declared and accessed in the simulator by name, making changes to BLOCK requires that the entire simulator be recompiled. Likewise, the primitive names are similarly "hard-coded" into the VOHL compiler in the file PRIMS; altering this file requires recompiling the VOHL compiler. This is obviously unacceptable, particularly for a CAD package intended for students in the laboratory, where frequent additions to (and subsequent deletions from) the primitive library are the rule.

The recompilation obstruction was circumvented by divorcing MultiSim's names for the primitives from the user's name for them. This is done by providing two name fields in each primitive description, one for use by the

system internally, and the other for the user's name for the cell. In addition to the fifteen provided primitives, an additional forty are declared but left with NULL descriptions until added later by the user. In this way, the programs can still address them by their (internal) names, their actual descriptions may be kept in separate data files until needed at runtime, and recompilation is not required for structure-only additions. The most visible manifestation of the new approach is the file PRIMITIV.DAT (Figure 7). This is a new data file containing five fields:

* the user-defined name for the primitive
* number of inputs
* number of outputs
* type of description available (0=block-level, 1=structural, 2=both)
* primitive level (to be discussed in Chapter 4)

```
15
READIN    0 0 2 0
AND       2 1 2 0
OR        2 1 2 0
NAND      2 1 2 0
NOR       2 1 2 0
INVERT    1 1 2 0
EXOR      2 1 2 0
ANDTHRE   3 1 2 0
NANDTHR   3 1 2 0
SRBLOCK   3 3 2 1
RETDBLO   6 5 2 1
ANDFOUR   4 1 2 0
NANDFOU   4 1 2 0
ORTHREE   3 1 2 0
ORFOUR    4 1 2 0
```

Figure 7. PRIMITIV.DAT file with sample primitives

As noted previously, there are two types of primitive descriptions (block and structural). However, in the interest of flexibility, it is no longer necessary for both to be available at the same time. It is possible to have library primitives described only by their VOHL structural code; circuits thus described will always be expanded. Likewise, primitives may be described in terms of their block behavior exclusively--however, it will not be possible to expand such primitives. Thus, there are now effectively three cases for primitive descriptions, corresponding to the method chosen for adding each primitive to the library:

The first consists of both the structure (in VOHL) and block behavior (C-language procedure with block delays); the fifteen original primitives are described in this fashion. This is the conventional form of cell addition, and is invoked by prefacing the module to be added with the keyword **ADDLIB**. The user is expected to supply the block-level C language procedure, and the structural definition is, of course, contained in the user's VOHL program. The compiler calculates the block delays itself.

In the second case, the user may not be interested in the block behavior of the circuit--it may be a critical component, and its behavior must be completely known in maximum detail at all times. For such a case, the block descriptions may be omitted; this process is called a structure-only addition. To perform this addition, the

module in the VOHL circuit to be added is prefaced with the keyword ADSTRUC.

Finally, if the circuit is very complex, or it is not necessary to completely specify every element of its behavior, the structural description may be omitted and the block description will always be used whenever reference to the circuit is made. This block-only addition is made by marking the circuit to be added with the keyword ADBLOCK.

C. STRUCTURE-ONLY ADDITION

Due to time constraints, the structure-only addition is the only method presently fully-installed into the compiler. The design considerations for block-only and complete addition may be found in Chapter 5, Future Research.

If structure-only additions are called for, the compiler opens a file and stores each new VOHL cell description there (several cells may be added at once, as long as each one is composed only of previously-defined primitives). It also strips off the addition keywords so that the circuit may be compiled normally. Once the circuit has been completely scanned for addition requests, compilation then proceeds in the usual manner. After compilation, the new cells' structural descriptions are added into the STRUC auxiliary file. Their user-defined names, number of inputs, number of outputs, and the "structure-only" flag are added into PRIMITIV.DAT for future reference.

Figure 8 illustrates a four-bit adder composed of single-bit full adders, which are in turn made up of half

adders. In this circuit, it is desired to add the half adder as a cell primitive, hence the ADSTRUC keyword placed just prior to the MODULE keyword of the half adder. Figure 9 shows the user circuit immediately prior to entering the actual compilation; the ADSTRUC keyword has been removed, and the half adder has been stored in the auxiliary file NEWCKTS.VHL. Once compilation is complete, the compiler enters NEWCKTS.VHL, detects a structural addition request from the keyword (which was left in place for this purpose), and makes the additions to PRIMITIV.DAT and STRUC (Figures 10 and 11 respectively).

D. USING THE ADDITIONS

Adding a structural description to the library is a straightforward process, but what is not so simple is making use of the cells thus added. Unless told to do otherwise with the EXPAND statement, MultiSim will use the block description for the primitives in the interest of speed. Due to the manner in which the compiler is built, it is not possible to intercept a call to a block procedure which is not present in the block library once compilation starts--the structural description must be available beforehand. To compound the problem, structure-only descriptions may be arbitrarily-many levels deep. For example, consider again the four-bit-adder of Figure 8. If both the full adder and half adder are in the primitive library, it is perfectly correct to describe the circuit as in Figure 12.

```
MODULE:  FOUR_BIT_ADDER;
INPUTS: A0, A1, A2, A3, B0, B1, B2, B3, CI0;
OUTPUTS: S0, S1, S2, S3, CO3;
TYPES: INTERNALS: CO0, CO1, CO2;
   {
   S0, CO0 = FULL_ADD(A0, B0, CI0);
   S1, CO1 = FULL_ADD(A1, B1, CO0);
   S2, CO2 = FULL_ADD(A2, B2, CO2);
   S3, CO3 = FULL_ADD(A3, B3, CO3);
   }

(simulation specifications )
(as desired                )

MODULE: FULL_ADD;
INPUTS: A, B, CI;
OUTPUTS: S, CO;
TYPES: INTERNALS: X, Y, Z;
   {
   X, Y = HALF_ADD(A, B);
   S, Z = HALF_ADD(X, CI);
   C0 = OR(Z, Y);
   }


ADSTRUC;
MODULE: HALF_ADD;
INPUTS: A, B;
OUTPUTS: S, CO;
TYPES: ;
   {
   S = EXOR(A, B);
   C0 = AND(A, B);
   }
END;
```

Figure 8. The Four-bit Adder
Prior to Structural Addition

```
MODULE:  FOUR_BIT_ADDER;
INPUTS: A0, A1, A2, A3, B0, B1, B2, B3, CI0;
OUTPUTS: S0, S1, S2, S3, CO3;
TYPES: INTERNALS: CO0, CO1, CO2;
   {
   S0, CO0 = FULL_ADD(A0, B0, CI0);
   S1, CO1 = FULL_ADD(A1, B1, CO0);
   S2, CO2 = FULL_ADD(A2, B2, CO2);
   S3, CO3 = FULL_ADD(A3, B3, CO3);
   }

(simulation specifications )
(as desired                )

MODULE: FULL_ADD;
INPUTS: A, B, CI;
OUTPUTS: S, CO;
TYPES: INTERNALS: X, Y, Z;
   {
   X, Y = HALF_ADD(A, B);
   S, Z = HALF_ADD(X, CI);
   C0 = OR(Z, Y);
   }

MODULE: HALF_ADD;
INPUTS: A, B;
OUTPUTS: S, CO;
TYPES: ;
   {
   S = EXOR(A, B);
   C0 = AND(A, B);
   }
END;
```

User's Circuit With Keyword Removed

```
ADSTRUC;
MODULE: HALFADD;
INPUTS: A, B;
OUTPUTS: S, CO;
TYPES: ;
   {
   S = EXOR(A, B);
   C0 = AND(A, B);
   }
END;
```

Contents of Auxiliary File NEWCKTS.VHL

Figure 9. New Module is Ready for Insertion Into Structure Library

```
16
READIN     0 0 2 0
AND        2 1 2 0
OR         2 1 2 0
NAND       2 1 2 0
NOR        2 1 2 0
INVERT     1 1 2 0
EXOR       2 1 2 0
ANDTHRE    3 1 2 0
NANDTHR    3 1 2 0
SRBLOCK    3 3 2 1
RETDBLO    6 5 2 1
ANDFOUR    4 1 2 0
NANDFOU    4 1 2 0
ORTHREE    3 1 2 0
ORFOUR     4 1 2 0
HALF_AD    2 2 1 1
```

Figure 10. PRIMITIV.DAT After the Addition

```
        .
        .
    existing
    modules
        .
        .
        .
    }
MODULE: HALF_ADD;
INPUTS: A, B;
OUTPUTS: S, CO;
TYPES: ;
    {
    S = EXOR(A, B);
    C0 = AND(A, B);
    }
```

Figure 11. Structural Description File After Addition

```
MODULE:  FOUR_BIT_ADDER;
INPUTS: A0, A1, A2, A3, B0, B1, B2, B3, CI0;
OUTPUTS: S0, S1, S2, S3, CO3;
TYPES: INTERNALS: CO0, CO1, CO2;
   {
   S0, CO0 = FULL_ADD(A0, B0, CI0);
   S1, CO1 = FULL_ADD(A1, B1, CO0);
   S2, CO2 = FULL_ADD(A2, B2, CO2);
   S3, CO3 = FULL_ADD(A3, B3, CO3);
   }

(simulation specifications )
(as desired                )
```

Figure 12. Four-Bit Adder
With Lower-level Cells as Primitives

Since neither the full adder or half adder have block descriptions, the structural descriptions must be made available as if an expansion were being performed on both submodules. This requirement led to a "precompiler" which examines the circuit prior to the VOHL compiler proper. It examines each function name, checking:

* first, to see if the function name corresponds to a valid primitive.

* second, if so, it inquires as to whether the structural description is the only one available for this primitive. If this is also the case, the precompiler passes control to a special "structure-only" expansion routine.

The expansion routine enters the primitive structural library and appends the description of the target primitive to the user's circuit; in a sense, it works very much like

the normal expansion procedure. However, unlike the normal one this procedure is recursive--it is possible, as in the case of the four bit adder, that the submodule just appended also contains function names with structure-only descriptions. The procedure must therefore examine the newly-added module in exactly the same fashion as the original circuit. If a module is found that fits the above two criteria, it simply calls itself and repeats the append process. The process stops when no function name fits the entry criteria.

Continuing the example, the expansion routine sniffs at the four-bit-adder of Figure 12 and notices that FULL_ADD has only a structural description. It therefore extracts the VOHL description of the FULL_ADDer and appends it to the end of the four bit adder circuit (Figure 13). (As the routine encounters subsequent references to FULL_ADD, it simply skips to the next line; only one copy of each submodule is necessary.) However, as the routine scans FULL_ADD, it likewise detects the HALF_ADDer, enters the primitive library and appends the structural description of the HALF_ADDer to the previous two cells. As the primitives that make up the half adder are completely described, the process may now stop leaving the circuit description of Figure 14 as the version to be compiled.

```
MODULE:  FOUR_BIT_ADDER;
INPUTS: A0, A1, A2, A3, B0, B1, B2, B3, CI0;
OUTPUTS: S0, S1, S2, S3, CO3;
TYPES: INTERNALS: CO0, CO1, CO2;
   {
   S0, CO0 = FULL_ADD(A0, B0, CI0);
   S1, CO1 = FULL_ADD(A1, B1, CO0);
   S2, CO2 = FULL_ADD(A2, B2, CO2);
   S3, CO3 = FULL_ADD(A3, B3, CO3);
   }

(simulation specifications )
(as desired                )

MODULE: FULL_ADD;
INPUTS: A, B, CI;
OUTPUTS: S, CO;
TYPES: INTERNALS: X, Y, Z;
   {
   X, Y = HALF_ADD(A, B);
   S, Z = HALF_ADD(X, CI);
   C0 = OR(Z, Y);
   }
```

Figure 13. The Adder After the Expansion Routine's First Pass

```
MODULE:  FOUR_BIT_ADDER;
INPUTS: A0, A1, A2, A3, B0, B1, B2, B3, CI0;
OUTPUTS: S0, S1, S2, S3, CO3;
TYPES: INTERNALS: CO0, CO1, CO2;
   {
   S0, CO0 = FULL_ADD(A0, B0, CI0);
   S1, CO1 = FULL_ADD(A1, B1, CO0);
   S2, CO2 = FULL_ADD(A2, B2, CO2);
   S3, CO3 = FULL_ADD(A3, B3, CO3);
   }

(simulation specifications )
(as desired                )

MODULE: FULL_ADD;
INPUTS: A, B, CI;
OUTPUTS: S, CO;
TYPES: INTERNALS: X, Y, Z;
   {
   X, Y = HALF_ADD(A, B);
   S, Z = HALF_ADD(X, CI);
   C0 = OR(Z, Y);
   }

MODULE: HALF_ADD;
INPUTS: A, B;
OUTPUTS: S, CO;
TYPES: ;
   {
   S = EXOR(A, B);
   C0 = AND(A, B);
   }
```

Figure 14. The Adder to be Passed to the Compiler

IV. ADDITIONAL ENHANCEMENTS

A. BUG FIXES

1. Hashing Algorithm

As more features are installed into a program and more of its capabilities are exercised, it is unfortunately quite natural that some previously-undetected bugs should crop up in the original version. To date, MultiSim has offered two.

The first was a minor bug in the hashing scheme that the compiler and simulator used to uniquely identify input and output variable names; the original hashing algorithm was a single-level procedure that computed the hashvalue by summing the ASCII codes of each character in the variable name. Though simple and usually effective, this scheme can produce collisions--for instance, the variables A2, B1 and C0 will all hash to the same value (A2 yields 65+50=115, B1 yields 66+49=115, C0 yields 67+48=115). The corrected code [Appendix B] stores the hashvalues in a small table and checks a newly-generated value against this table. If a match is found (e.g., two variable names hash to the same value), a prime number is added to the newly-produced hashvalue and the table comparison restarts. This process repeats until no collision occurs, thus assuring that no two variable names will produce the identical hashvalue.

2. Multilevel Expansion

The second bug was a bit more interesting. As originally conceived, the EXPAND keyword was to enable the user to call for a primitive to be expanded to any lower level [Ref. 5]. As installed, however, issuing the EXPAND will cause the primitive specified to be expanded only to the next lower level (see also Chapter 3). In order for multilevel expansion to occur, the compiler needs to check each submodule to see if:

* the level called for has been reached, or;
* the expansion has been carried out to the depth requested, but the named target primitive is not contained in any submodule.

Further, the original library primitives were not ordered in relation to each other; if, for instance, a full adder were to be expanded into its constituent gates, there was no way to tell how many levels deep the expansion must go.

The first step in implementing true multilevel expansion was to add a "level" field to each primitive description (Figure 7 of Chapter 4). The level field operates quite simply:

Basic gates, the lowest-level primitives, are assigned level 0. Any element composed of level-0 gates is therefore level 1. If a circuit is composed of at least one level-1 element it is assigned level 2, and so on. Thus, the "level" field indicates where in the hierarchy each primitive occurs.

A recursive procedure [Appendix B] was built that checks the constituent elements of a primitive to be expanded. If the "destination" primitive is of a lower level than a particular submodule, this submodule will be expanded and the submodules that make it up will also be checked. The process repeats until either all occurrences of the "destination" have been found or all primitives of the same level as the destination are reached. If the latter occurs, the compiler alerts the user that it couldn't find the destination and provides the option to either continue anyway or to quit.

Consider the four-bit adder of Figure 15, and allow the full adder and half adder to be completely-described (e.g., both structural and block-only descriptions in the appropriate libraries). In this case, it is desired to expand the full adders into AND gates. Figure 16 shows the full adder and half adder for reference.

As the compiler encounters the EXPAND: FULLADD: AND; statement, it adds the full adder to the expand table and enters the recursive routine. The routine examines the three circuit lines of the full adder and discovers that two of them are of a higher level than the AND gate (the two HALFADD lines). Therefore the routine adds the half adder to the expand table and calls itself again to examine the half adder. The half adder contains an AND gate, and since

```
MODULE : ADDER4;
INPUTS : A0, A1, A2, A3, B0, B1, B2, B3, CI0;
OUTPUTS: S0, S1, S2, S3, CO3;
TYPES : INTERNALS : CO0, CO1, CO2;
   {
   S0, CO0=FULLADD(A0, B0, CI0);
   S1, CO1=FULLADD(A1, B1, CO0);
   S2, CO2=FULLADD(A2, B2, CO1);
   S3, CO3=FULLADD(A3, B3, CO2);
   }

DEFINE:;
EXPAND: FULLADD: AND;
INTIALIZE: CO0=0, CO1=0, CO2=0;
PRINTOUT: S3, S2, S1, S0;
END;
```

Figure 15. The Four-Bit Adder Showing Multilevel Expansion

```
MODULE : FULLADD;
INPUTS : A, B, CIN;
OUTPUTS : S, C0;
TYPES : INTERNALS: X, Y, Z;
   {
   X, Y=HALFADD(A, B);
   S, Z=HALFADD(X, CIN);
   C0=OR(Z, Y);
   }

MODULE : HALFADD;
INPUTS : A, B;
OUTPUTS: S, C0;
TYPES: ;
   {
   S=EXOR(A, B);
   C0=AND(A,B);
   }
```

Figure 16. Lower-level Components

the EXOR is also a level-0 gate the routine exits and marks the search for ANDs as successful. Thus, after the routine exits the expand table contains not only the full adder, but the half adder as well, and the full adder will in fact be expanded into AND gates in the process described in Chapter 3.

B. NEW FEATURES

1. Primitive Cell Substitution

When developing VLSI circuits it is not uncommon to integrate devices spanning several technologies. An example of this is a RAM (Random-Access read/write Memory) module--the memory cells will most likely be composed of CMOS or NMOS, but the "housekeeping" circuits will in all probability be TTL. Each technology has its strengths and weaknesses, and the circuit designer may frequently be interested in evaluating the effects and tradeoffs of each. In addition, the phenomenon of skew is of vital concern in many combinational as well as sequential circuits--here the designer is interested in the effects of varying delays along parallel circuit paths. Such delays may be caused by gates whose propagation delays are nonuniform from gate to gate; their effect is to cause signals which would ordinarily arrive at a given point at the same time to differ slightly, with occasionally serious consequences.

In an effort to model such phenomena, the original MultiSim provided the EXPAND and DEFINE keywords to allow a

user to specify individual parameters for the various circuits composing the design. However, such definitions are global; issuing a DEFINE for an AND gate, for instance, causes all AND gates appearing in the circuit to possess the defined characteristics. This poses no difficulty if the user has built all of the modules and submodules; the TYPES statement may be used to create a particular version of a primitive and the DEFINE statement issued for the new version (in the case of the AND, creating an AND2 of type AND and DEFINEing the AND2).

Unfortunately, there is still the matter of any higher-level primitives in the cell library. Consider again the four-bit adder of Figure 15--if the adder is made a structural primitive, then there is no way to change the characteristics of the AND gate in the half-adder short of issuing a DEFINE statement for the AND. Note that this will have the effect of specifying parameters for all AND gates in the entire circuit whether that was the intended effect or not. For example, to model skewing in a sixteen-bit adder it is most useful to specify four different types of AND gates (one for the AND gate in each four-bit adder), substitute these user-defined gates for the ones contained in the primitive definition and observe their effects on the output.

Thus was born the USING clause. The USING keyword and its associated parameter list preface each VOHL circuit

line where such a substitution is desired. In the four-bit adder of Figure 17, the designer has altered the previous design to specify a different EXOR and AND gate from the defaults in two of the full-adders; it is suspected that skewing might occur in the outputs of these two adders and the effects of such skewing need to be determined. (As before, in this example the full and half adders are assumed to be fully-defined primitives.)

When the compiler is invoked, the circuit is copied to a temporary file and any circuit lines with **USING** clauses are delimited by special keywords (Figure 18). The user-defined **TYPES** are saved in a special table, as are the parameters to the **USING** clause. The level of the first parameter in the **USING** clause is checked against the function name in the circuit line; if the function name is a primitive, and further, if the function is of higher level, the clause handler issues an expansion request. Here is revealed the greatest power of the **USING** statement--it is possible to modify existing primitives at will, and the process is completely transparent to the user.

This amended circuit is then passed to the original compiler to be expanded into a single level as described previously; however, the expanded code corresponding to a line containing a USING will still be marked by the delimiters (Figure 19). Note that although the first and third lines of the original circuit, each containing a full

```
MODULE : ADDER4;
INPUTS : A0, A1, A2, A3, B0, B1, B2, B3, CI0;
OUTPUTS: S0, S1, S2, S3, CO3;
TYPES : EXOR : EXOR1, EXOR2;
        OR : OR1, OR2;
        INTERNALS : CO0, CO1, CO2;

   {
   USING(NOEXP, EXOR1, OR1): S0, CO0=FULLADD(A0, B0, CI0);
   S1, CO1=FULLADD(A1, B1, CO0);
   USING(NOEXP, EXOR2, OR2): S2, CO2=FULLADD(A2, B2, CO1);
   S3, CO3=FULLADD(A3, B3, CO2);
   }

DEFINE: EXOR1: RISEDELAY(0,0)=2,
               FALLDELAY(0,0)=4;
        EXOR2: RISEDELAY(0,0)=3;
               FALLDELAY(0,0)=3;
        OR1 :  RISEDELAY(1,0)=2;
        OR2 :  RISEDELAY(1,0)=3;
INITIALIZE: CO0=0; CO1=0; CO2=0;
PRINTOUT: S0, S1, S2, S3, CO3;
```

Figure 17.
The Four-bit Adder With USING

adder, have been expanded into the low-level interccnnections composing the full adder, the two "lines" are still detectable by merely sensing the delimiters.

This is, of course, precisely what the USING handler does. Once the "start" delimiter is detected, each function name encountered is compared to compared to the list of TYPES made earlier. If there is a match, the parameter list of the USING clause is compared to the user-defined TYPE. If one of these also matches, the user-defined function thus found is substituted for the existing one.

```
MODULE : ADDER4;
INPUTS : A0, A1, A2, A3, B0, B1, B2, B3, CI0;
OUTPUTS: S0, S1, S2, S3, CO3;
TYPES : EXOR : EXOR1, EXOR2;
        OR : OR1, OR2;
        INTERNALS : CO0, CO1, CO2;

   {
SWAPLIN;
   S0, CO0=FULLADD(A0, B0, CI0);
ENDSWAP;
   S1, CO1=FULLADD(A1, B1, CO0);
SWAPLIN;
   S2, CO2=FULLADD(A2, B2, CO1);
ENDSWAP;
   S3, CO3=FULLADD(A3, B3, CO2);
   }

DEFINE: EXOR1: RISEDELAY(0,0)=2,
               FALLDELAY(0,0)=4;
        EXOR2: RISEDELAY(0,0)=3;
               FALLDELAY(0,0)=3;
        OR1 :  RISEDELAY(1,0)=2;
        OR2 :  RISEDELAY(1,0)=3;
INITIALIZE: CO0=0; CO1=0; C02=0;
PRINTOUT: S0, S1, S2, S3, C03;

MODULE : FULLADD;
INPUTS : A, B, CIN;
OUTPUTS : S, C0;
TYPES : INTERNALS: X, Y, Z;
   {
   X, Y=HALFADD(A, B);
   S, Z=HALFADD(X, CIN);
   C0=OR(Z, Y);
   }

MODULE : HALFADD;
INPUTS : A, B;
OUTPUTS: S, C0;
TYPES: ;
   {
   S=EXOR(A, B);
   C0=AND(A,B);
   }
```

Figure 18. The Adder
After Delimiters Installed

```
MODULE : ADDER4;
INPUTS : A0, A1, A2, A3, B0, B1, B2, B3, CI0;
OUTPUTS: S0, S1, S2, S3, CO3;
TYPES: EXOR: EXOR1, EXOR2;
       OR : OR1, OR2;
       INTERNALS: CO0, CO1, CO2;
       INTERNALS: X0, X1, X2, X3, Y0, Y1, Y2, Y3;
   {
SWAPLIN;
   X0=EXOR(A0, B0);
   Y0=AND(A0, B0);
   S0=EXOR(X0, CI0);
   Z0=AND(X0, CI0);
   CO0=OR(Z0, Y0);
ENDSWAP;
   X1=EXOR(A1, B1);
   Y1=AND(A1, B1);
   S1=EXOR(X1, CO0);
   Z1=AND(X1, CO0);
   CO1=OR(Z1, Y1);
SWAPLIN;
   X2=EXOR(A2, B2);
   Y2=AND(A2, B2);
   S2=EXOR(X2, CO1);
   Z2=AND(X2, CO1);
   CO2=OR(Z2, Y2);
ENDSWAP;
   X3=EXOR(A3, B3);
   Y3=AND(A3, B3);
   S3=EXOR(X3, CO2);
   Z3=AND(X3, CO2);
   CO3=OR(Z3, Y3);
   }

DEFINE: EXOR1: RISEDELAY(0,0)=2,
               FALLDELAY(0,0)=4;
        EXOR2: RISEDELAY(0,0)=3;
               FALLDELAY(0,0)=3;
        OR1 :  RISEDELAY(1,0)=2;
        OR2 :  RISEDELAY(1,0)=3;
INITIALIZE: CO0=0; CO1=0; C02=0;
PRINTOUT: S0, S1, S2, S3, C03;
```

Figure 19. The Adder After Expansion and Prior to Substitution

The process then repeats for each circuit line until the "end" delimiter is encountered.

Returning to the example, the designer wishes to model the effects of nonuniformities in the EXOR and OR gates in two of the half adders. To begin, two types of EXOR and two types of OR are specified in the **TYPES** line, with the parameters to be perturbed set in the **DEFINE** line. Since the EXOR is the "deepest" level that the user wishes to look, it is the first parameter to the **USING** clause.

The first variation on the EXOR and the OR will go into the first full adder. Thus, the syntax for the first using clause and statement line is:

```
USING(EXOR1, OR1):  S0, CI0 = FULLADD(A0, B0, CI0);
```

Likewise, the form of the second statement line is:

```
USING(EXOR2, OR2):  S2, CO1 = FULLADD(A2, B2, CO0);
```

When this VOHL circuit is passed to the compiler, the **USING** keyword causes control to be passed to the clause handler. As the handler senses the first group of candidate lines for substitution, the first EXOR in the expanded circuit of Figure 19 is compared against the type list. There is a match since an EXOR has been defined. The two types of EXOR are then compared to the **USING** parameter list, generating a match on EXOR1. Since a both the proper function type and the proper parameter have been found,

EXOR1 is then substituted for EXOR in the VOHL circuit description. There is no match with the AND in the next circuit line, as no ANDs have been defined. The EXOR in the third line causes a repeat of the above sequence, resulting in another EXOR1 substitution. Likewise, the OR on the fifth line generates a match in the type list and OR1 matches the other parameter to the **USING** clause, so OR1 is substituted for OR in the circuit. After the OR, **ENDSWAP** tells the handler to ignore the next several lines until the process begins again on the **SWAPLIN** keyword. The handler terminates upon sensing the closing brace which marks the end of the circuit.

After the process is complete, the circuit will appear as in Figure 20; it is this version that will actually be compiled for simulation. Note that there are now three types of EXOR and three types of OR in the VOHL circuit, all with different parameters. Further, it was not necessary to build three different types of half-adder, in whose descriptions the substitutions were actually performed; rather, it is only necessary to develop one submodule and change its components with the USING clause. One may then quickly observe that this feature adds a significant degree of flexibility and power to the VOHL description language.

Of course, the addition routines operate concurrently with module substitution; however, the original

```
MODULE : ADDER4;
INPUTS : A0, A1, A2, A3, B0, B1, B2, B3, CI0;
OUTPUTS: S0, S1, S2, S3, CO3;
TYPES: EXOR: EXOR1, EXOR2;
       OR : OR1, OR2;
       INTERNALS: CO0, CO1, CO2;
       INTERNALS: X0, X1, X2, X3, Y0, Y1, Y2, Y3;
   {
   X0=EXOR1(A0, B0);
   Y0=AND(A0, B0);
   S0=EXOR1(X0, CI0);
   Z0=AND(X0, CI0);
   CO0=OR1(Z0, Y0);
   X1=EXOR(A1, B1);
   Y1=AND(A1, B1);
   S1=EXOR(X1, CO0);
   Z1=AND(X1, CO0);
   CO1=OR(Z1, Y1);
   X2=EXOR2(A2, B2);
   Y2=AND(A2, B2);
   S2=EXOR2(X2, CO1);
   Z2=AND(X2, CO1);
   CO2=OR2(Z2, Y2);
   X3=EXOR(A3, B3);
   Y3=AND(A3, B3);
   S3=EXOR(X3, CO2);
   Z3=AND(X3, CO2);
   CO3=OR(Z3, Y3);
   }

DEFINE: EXOR1: RISEDELAY(0,0)=2,
               FALLDELAY(0,0)=4;
        EXOR2: RISEDELAY(0,0)=3;
               FALLDELAY(0,0)=3;
        OR1 :  RISEDELAY(1,0)=2;
        OR2 :  RISEDELAY(1,0)=3;
INITIALIZE: CO0=0; CO1=0; C02=0;
PRINTOUT: S0, S1, S2, S3, C03;
```

Figure 20. The Adder
After Substitution

(e.g., as if the USING were not present) VOHL circuit is the one that will be saved as a library primitive.

2. Addition Without Compilation

It is occasionally useful, particularly in a laboratory setting, to add a circuit or group of circuits to the cell library without compiling or simulating them. The user's prime interest here is in building up the library or perhaps customizing it to a particular purpose. To accomodate this, the precompiler, which senses the addition keys, was modified slightly to check for the absence of simulation control specifications when the first circuit module is preceded by an addition keyword. If this happens, compilation is discontinued and control passes immediately to the library addition routines. Also, since more than one module may be added at a time, this technique may be used to quickly add large numbers of circuits to the primitive library.

Consider the circuit of Figure 21. The ADSTRUC keywords before each module indicate to the compiler that both the RSCELL and the JK flip-flop are to be added to the VOHL structure library. The ADSTRUC before the first module (RSCELL in the example) causes a flag to be set; this flag is tested when the second ADSTRUC is encountered instead of the DEFINE that would normally occur were the simulation specs present. Since the flag is set, the compiler concludes that no compilation is necessary, enters the

structure-only addition routine and inserts both modules into the appropriate libraries as discussed in Chapter 4. Note that the RSCELL must be added prior to the JKFF; this is so that the primitive levels may be calculated properly. The rules for module addition discussed in Chapter 4 are equally valid here--new modules must be built out of only previously-entered modules.

```
ADSTRUC ;
MODULE : RSCELL ;
INPUTS : S, R ;
OUTPUTS: Q, QC ;
TYPES : ;
   {
   Q = NAND(S, QC);
   QC = NAND(R, Q);
   }

ADSTRUC ;
MODULE : JKFF ;
INPUTS : J, K, CLK;
OUTPUTS: Q, QBAR;
TYPES : INTERNALS : S1, R1;
   {
   S1 = NANDTHR(QBAR, J, CLK);
   R1 = NANDTHR(CLK, Q, K);
   Q, QBAR = RSCELL(S1, R1);
   }

END ;
```

Figure 21. The JK Flip-Flop Addition Without Compilation

V. CONCLUSIONS AND FUTURE RESEARCH

A. WAYPOINTS

It is perhaps characteristic of projects of this magnitude that they are never quite finished, no matter how far the boundaries have been pressed out nor how many artisans have had a hand in them. So it is with MultiSim. The extensions to MultiSim described in the preceding chapters bring it to the verge of the student laboratory and even the marketplace, but there is yet work to be done:

* The block-level and "complete" (block and structural together) additions must be completed and installed.

* A way must be found to circumvent the requirement for recompiling components of MultiSim, or such recompilations must at the very least be made transparent to the user.

* An interface must be devised and constructed to allow MultiSim to be used with the EE Designer schematic drawing and capture program. Specifically, this includes a DOS shell to accomodate both programs and their auxiliary files, a netlist translator to convert EE Designer netlists into VOHL, and allowing EE Designer to import the simulation results from MultiSim's timing wheel.

1. Block-level Addition

As may be observed from Chapter 4, adding the structural representation of a VOHL circuit to the primitive library is a reasonably straightforward process. The VOHL code to be added is extracted from the circuit being modeled; the inputs and outputs are counted; finally these along with the module name and a flag marking the new module as "Structure-only" are stored in PRIMITIV.DAT.

Block-only additions are somewhat more complicated as they have more to do with the simulator than the compiler. As discussed in Chapter 3, the block-level behavioral descriptions for each primitive are C-language procedures contained in the auxiliary file BLOCK; this file, along with the declarations in FADDR (Figure 22) and pointer initializations in FTYPE (Figure 23) are included into the simulator when the simulator is compiled on the host machine. In the present version of the simulator, adding a block-level primitive requires the following steps:

* adding the new primitive's function declaration to FTYPE
* adding the pointer initialization to FADDR and incrementing the primitive count also contained there
* determining the propagation delays from each input of the new circuit to each output, then entering these values into the block delay data file BLDEL.
* appending the new block-level C code to BLOCK, then recompiling the entire timing wheel simulator.

For instance, to add the half adder of previous discussions to the block library, the line

```
int HALFADD();
```

must be added to FTYPE. Likewise, a pointer to the new function contained in the array pnfn[] is initialized with the statement

```
pnfn[15] = HALFADD;
```

```
int NAND();
int NOR();
int ANDTHRE();
int ORTHREE();
int NANDTHR();
int SRBLOCK();
int RETDBLO();
int ANDFOUR();
int ORFOUR();
int EXOR();
```

Figure 22. The Auxiliary File FTYPE

```
pnfn[3] = NAND;
pnfn[4] = NOR;
pnfn[6] = EXOR;
pnfn[7] = ANDTHRE;
pnfn[8] = NANDTHR;
pnfn[9] = SRBLOCK;
pnfn[10]= RETDBLO;
pnfn[11]= ANDFOUR;
pnfn[12]= NANDFOU;
pnfn[13]= ORTHREE;
pnfn[14]= ORFOUR;
pncnt = 15;
```

Figure 23. The Auxiliary File FADDR

which must be added to FADDR. The last line of FADDR is changed to read

```
pncnt = 16;
```

Lastly, the C-language code for the half adder is appended to BLOCK after which the timing wheel is compiled using the appropriate commands (LC and LINK on the IBM PC/AT, and cc on the VAX).

This procedure is not amenable to easy installation of new primitives. What is propcsed is a method similar to that used on the compiler:

* FTYPE will be extended to declare an additional set of primitives (the VOHL compiler uses an additional forty). These will be internal names transparent to the user. The new FTYPE, which will remain an #include file, is shown in Figure 24.

* FADDR will be made into an external procedure, independent of the timing wheel. The additional primitive declarations will be assigned NULL pointers until an actual description is added. When this occurs, the system name of the new primitive will replace the NULL (Figure 25).

* Likewise, the procedures that compose the BLOCK file will all be made external; new primitive C-language block descriptions will still be placed in this file.

The advantage to converting the #include files into external procedures is that only the external procedures need to be recompiled when they are altered. Since the two files concerned are much smaller than the timing wheel itself, compilation will proceed much more quickly. In addition, the MSDOS linker teamed with a hard disk works very swiftly as well; the total time to recompile and relink

```
int NAND();
int NOR();
int ANDTHRE();
int ORTHREE();
int NANDTHR();
int SRBLOCK();
int RETDBLO();
int ANDFOUR();
int ORFOUR();
int EXOR();
int USER1();
int USER2();
int USER3();
     .
     .
int USERn();
```

Figure 24. The Auxiliary File FTYPE Modified for New Primitive Declarations

```
pnfn[3] = NAND;
pnfn[4] = NOR;
pnfn[6] = EXOR;
pnfn[7] = ANDTHRE;
pnfn[8] = NANDTHR;
pnfn[9] = SRBLOCK;
pnfn[10]= RETDBLO;
pnfn[11]= ANDFOUR;
pnfn[12]= NANDFOU;
pnfn[13]= ORTHREE;
pnfn[14]= ORFOUR;
pnfn[15]= USER1;
pnfn[16]= USER2;
pnfn[17]= NULL;
pnfn[18]= NULL;
     .
     .
pnfn[55]= NULL;
pncnt = 17;
```

Figure 25. The External Procedure File FADDR With Two User-Defined Primitives Installed

the new BLOCK and FADDR files will typically be less than ninety seconds. In fact, a shell can be built (or added to the existing one) specifically to handle block-level additions--in this way the inconvenience to the user will be minimized since the procedure is automatic.

The operation of the new mechanism is as follows:

* In addition to the VOHL circuit to be modeled, the user places block-level code for the primitive to be added into a separate ASCII text file. When the compiler senses an ADBLOCK keyword in the VOHL circuit, this text file will be entered. Using the two name fields of PRIMITIV.DAT, the compiler matches the user's name for the new primitive to its system (USERnn) name and passes control to the block-level addition routine.

* The addition routine enters FADDR and adds the

  pnfn[primitive number] = USERnn;

  line, as well as updates the primitive count pncnt. It then adds the user-provided block code to BLOCK, substituting the system name for the user's name (necessary since although procedures must be declared in advance, the routine obviously has no information as to what the user may choose to name a primitive).

* The block delays are tabulated and saved into BLDEL.

* A flag is set, which upon exiting the VOHL compiler and simulator will invoke the Lattice C compiler. FADDR and BLOCK are recompiled and relinked with the timing wheel, completing the installation of the block-level primitive.

The most difficult part of the block-level addition is the calculation of block delays. Dr. Mahmood has provided an algorithm [Ref. 6, pp. 108-126], but unfortunately time expired before it could be implemented as part of the work this thesis describes. The remainder of

the process is actually quite simple and in many ways is the same as for the structure-only addition.

Finally, "complete" addition combines the structural and block-level addition procedures to install a completely-described primitive.

2. EE Designer Interface

The **EE** Designer commercial CAD program is presently available for the PC/AT's in the Computer Design laboratory. As EE Designer produces netlists it is possible to use this program for schematic drawing and capture, subsequently producing a circuit netlist to be passed to MultiSim for compilation and simulation (substituting MultiSim's extremely fast simulator for EE Designer's slow one). However, since EE Designer produces its netlists in terms of standard part numbers (i.e., 7400 for a quad two-input NAND) it is necessary to translate these netlists into VOHL. This also should be a relatively straightforward process since the VOHL primitives correspond to standard parts. The major concern is the efficiency and speed of the translation--if the translation algorithm is sloppy, much of the speed advantage of MultiSim will be lost. Consequently, considerable effort and care must be devoted to this algorithm. The reward for smooth translation is simultaneously obtaining the graphics capabilities of EE Designer while enjoying the speed of MultiSim's simulator.

B. CONCLUSION AND CLOSING REMARKS

MultiSim on a desktop microcomputer brings considerable power to the circuit designer; further, it does so at a price that was heretofore unattainable. IBM PC-type machines are virtually standard equipment in most offices, and AT-class machines are rapidly becoming as common--the ability to quickly simulate large circuits on such machines offers considerable cost savings to both schools and corporations. Perhaps the greatest benefit is to schools, since their budgets are the tightest of the users of CAD equipment and the circuits they simulate generally do not require the computing muscle of a VAX-class machine. To be sure, MultiSim does have its limits--depending on the memory available, a maximum of 1500-2000 elements can be modeled (several times this amount on MC680x0-based machines such as the Commodore Amiga), but there are numerous applications where this is perfectly adequate.

MultiSim offers an impressive mix of performance, particularly in the speed of its event-driven simulator, versus price of its typical host machines ($6000 and less for IBM PC/AT-class computers). The enhancements to MultiSim described in this thesis make it a powerful, flexible tool for VLSI circuit design. For both designer and educator, it is worthy of careful consideration.

LIST OF REFERENCES

1. Sakallah, Karem and Director, Stephen, "SAMSON2: An Event Driven VLSI Circuit Simulator," IEEE Transactions on Computer Aided Design, Vol. CAD-4, No. 4, 1985.

2. DeWilde, P.M., et al., "Switch Level Timing Simulation," IEEE 1985 International Conference on Computer-Aided Design, IEEE Computer Society Press, 1985.

3. Goering, Richard, "Simulation Challenges Breadboarding for Design Verification," Computer Design, June 15, 1986.

4. Miller, D. and Miranker, G., "Terminal-based Engineering System Cuts Logic-Design Time Tenfold," Electronics, Vol. 55, No. 18, 1982.

5. Ulrich, Ernst, "Exclusive Simulation of Activity in Digital Networks," Communications of the ACM, February, 1969.

6. Mahmood, Ausif, "Development of a Multilevel Logic Simulator for VLSI Systems," PhD. Dissertation, Washington State University, August, 1985.

APPENDIX A.
QUICK REFERENCE CARDS

1. VOHL QUICK REFERENCE CARD

Keywords and their uses:

MODULE — The MODULE keyword identifies the beginning of a new circuit or subcircuit. The name of the module immediately follows the MODULE keyword and a colon separates the two. A semicolon follows the module name.

```
MODULE : HALFADD ;
```

INPUTS — This keyword identifies the inputs to the circuit. Inputs are listed after the INPUTS keyword and separated by commas; white spaces are optional. The last input is followed by a semicolon. If there are more inputs than will fit on a single line multiple lines may be used--terminate the last input in each line with a comma instead of a semicolon. As before, the final input should be followed by the semicolon.

```
INPUTS : A, B ;
```

OUTPUTS — Likewise, this keyword identifies the outputs from the circuit and are listed after OUTPUTS, separated by commas. As with INPUTS, if more than one line is needed simply terminate the last keyword in the line with a comma.

```
OUTPUTS: S, CO ;
```

TYPES — The TYPES statement is used for two purposes: First, it allows the user to specify variations of the standard primitives; and second, it is used for the creation of internal scratchpad variables. If, for instance, it is desired to construct a circuit out of a different NAND gate than the one available in the library, the user issues the line:

```
TYPES : NAND : NAND1 ;
```

and may then construct a circuit out of NAND1s. This use of the TYPES keyword also supports the USING keyword discussed below (where it is desired to produce multiple variations of previously-defined library primitives). To create scratchpad variables, the TYPES statement is followed by the INTERNALS keyword; the scratchpad variables follow INTERNALS.

```
TYPES : INTERNALS : DUMMY1, DUMMY2, ..;
```

Note: if the first form of the TYPES line has already been used and it is also desired to declare INTERNALS, the form is as follows:

```
TYPES : NAND : NAND1 ;
        INTERNALS : DUMMY1, DUMMY2, ..;
```

(The INTERNALS doesn't need a second TYPES keyword.)

The VOHL circuit is composed of individual circuit lines which resemble algebraic statements. A circuit line is composed of an input list, a function name, and an output list. The form is:

```
outputs = function(inputs) ;
```

For example, a statement line using the HALFADDer would appear as:

```
S0, CARRY0 = HALFADD(A0, B0) ;
```

The circuit body is enclosed by braces so that the completed circuit appears as:

```
{         (open brace=start of circuit)
theseoutputs = function(theseinputs) ;
thoseoutputs = function(thoseinputs) ;
otheroutputs = function(otherinputs) ;
       .
       .
 }         (close brace=end of circuit)
```

DEFINE This is used to specify particular parameters for a primitive. The user may specify such parameters as RISEDELAY,

FALLDELAY, FANOUT, and TECHNOLOGY. The RISEDELAY and FALLDELAY keywords require the number of the input and the number of the output for which the parameter applies. It is not necessary to specify all of the parameters when using DEFINE, merely the ones it is desired to change.

```
DEFINE : NAND : RISEDELAY(0,0)=3,  (1st input->1st output)
                FALLDELAY(1,0)=2,  (2nd input->1st output)
                FANOUT= 10,
                TECHNOLOGY=TTL;
```

EXPAND This keyword is used when a greater level of detail is required than the block-level description. The primitive to be expanded follows the EXPAND, which is in turn followed by the level the expansion is to be carried to. If desired, definition parameters (though without the DEFINE) may be placed after the primitive the expansion will be carried to. If just the expansion is desired, the statement appears as:

```
EXPAND : HALFADD : EXOR;
```

If the user wishes to specify parameters for the EXOR, the syntax will be:

```
EXPAND : HALFADD : EXOR : RISEDELAY(0,0)=3,
                          etc.  ;
```

INITIAL The INITIAL (or INITIALIZE) keyword is used to create the initial conditions for the simulation. Following the INITIAL keyword, the variables are listed along with the value (0, 1 or 2) that is to be assigned to them as the simulation begins.

```
INITIAL: A0=0, A1=1, B0=0, . . ;
```

PRINTOUT Finally, the list of variables the user wishes to observe are indicated here, separated by commas.

```
PRINTOUT: X0, X1, X2, X3, . . ;
```

END This is the very last line in the VOHL

circuit, and occurs after all modules and submodules. A semicolon follows the END keyword.

USING — The USING clause appears in the circuit description prior to the output list and contains a list of primitives to be substituted for those normally occuring. Each element in the USING parameter list should appear in the TYPES declarations, and it may also appear in the DEFINE statement if the user wishes to modify any parameters. Whereas the EXPAND and DEFINE statements together allow only one version of a primitive, USING allows as many different versions as the user requires. For instance, consider a circuit with three half-adders; for whatever reason it is necessary to provide for three **different** half-adders composed of different types of EXOR gates. With EXPAND and DEFINE, this would not be possible unless the user built three half-adders with the different EXORs. If the half-adder is a system primitive, all that is necessary is:

```
TYPES : EXOR : EXOR1, EXOR2, EXOR3 ;
   {
        .
        .
   USING(EXOR1): S1, CARRY1 = HALFADD(A1, B1);
   USING(EXOR2): S2, CARRY2 = HALFADD(A2, B2);
   USING(EXOR3): S3, CARRY3 = HALFADD(A3, B3);
        .
        .
   }
DEFINE: EXOR1: (parameters);
        EXOR2: (parameters);
        EXOR3: (parameters);
```

ADSTRUC — Finally, this keyword is used to add a module into the primitive structural library. It is followed by a semicolon and precedes the MODULE line of the circuit to be added.

Notes:

* It is not necessary to DEFINE parameters, however, the keyword must appear. In no parameters are to be DEFINEd, issue a:

  DEFINE : ;

  The EXPAND keyword does not appear at all if it is not needed.

* With the present version of the code generator, it is necessary to INITIALIZE at least one variable. If this is not done the compiler gets angry.

* Simulation control specifications (DEFINE, INITIALIZE, PRINTOUT, etc.) must only appear after the first module. Simulation specifications on submodules will be disregarded and will in fact cause a compilation error.

* If all that is desired is to add a module to the structural library, simply issue an ADSTRUC; on the line prior to the MODULE line and do not include simulation specifications.

2. DOS SHELL QUICK REFERENCE CARD

The MultiSim package is located in the \SIML directory of each IBM PC/AT in the Computer Design Laboratory. To enter this directory, type

```
cd \siml
```

at the MSDOS prompt. Once there, type

```
path2
```

to set up the MSDOS directory search paths for the various support programs. The KEDIT editor may be used to build VOHL circuits for modeling. The circuit may possess a name of up to eight characters long, and an extension should not be used for the circuit to be modeled. It is good practice to save a backup copy of the circuit; a convenient method is to use the same name but with a .ckt extension added. It will also be necessary to produce a file of input data for the circuit; KEDIT may be used for this purpose also. The input data should be of the same name as the circuit, but with a .in extension. For example, the four bit adder cn the PC/AT is called add4, with a backup version in add4.ckt and inputs in add4.in.

The MultiSim package is invoked with one of two commands: MODEL, which is the original, unenhanced MultiSim; or MODEL2, which includes the features described in the thesis. For all but those concerned with installing further features into MultiSim, MODEL may be disregarded (though its operation is identical with MODEL2).

There are three options with MODEL2:

C: Compile but do not simulate. This is used to check for VOHL syntax errors. If the compilation is successful the results will be saved to the hard disk. No circuit input file is necessary for compilation-only.

S: Go directly to simulation. The circuit must have been previously compiled and an input file must be available.

E: End-to-end. MODEL2 will call the compiler, then immediately enter the simulator. As with the Simulate case, an input file must be available.

The syntax for the MODEL2 command line is:

    model2 <circuitname> <option>

where <circuitname> is the name of the file to be operated on and <option> is either E, S, or C. Thus, to both compile and simulate the four-bit adder in a single operation, the command is:

    model2 add4 e

When a simulation is performed MODEL2 will display the results on the screen, then save them to the hard disk in the file <circuitname>.out; thus the results of the four-bit adder simulation will be in add4.out.

3. UNIX VAX QUICK REFERENCE CARD

All of the enhancements installed into the microcomputer version of MultiSim are also available in the VAX version, but there is no shell on the VAX corresponding to MODEL2. Consequently, the compiler and simulator must be invoked directly from the UNIX prompt. Because of this, the MSDOS naming conventions do not apply--circuits and data input files may have arbitrary and unrelated names (it is still good practice, however, to maintain the convention from MSDOS to assist in keeping track of files).

The VOHL compiler is contained in the executable file cadd while the simulator is in the executable file sim. To use the compiler, type

```
cadd <filename>
```

where <filename> contains the circuit to be compiled for simulation. To enter the simulator, type

```
sim <input file> <output file>
```

where <input file> contains the input data for the circuit and <output file> is the intended destination of the simulation results. As with the MSDOS version, the circuit must be compiled before it can be simulated; the compiler output is in the data file simdata, and this file must be present before the simulator can be invoked.

APPENDIX B.

VOHL COMPILER SOURCE CODE

CADD.C — The main module for the VOHL compiler, including the code generator.

PRECOMP.C — The precompilation routines, including the USING clause handler, the structure-only module handler, the module counter and a keyword stripper. This group of procedures contains the bulk of the work described in the thesis.

PRIMS.C — The primitive initialization routine converted into an external procedure.

```
/*****************************************************************
 *                                                               *
 *             Parsing and Code Generation Program               *
 *                         for the                               *
 *                    Multilevel Simulator                       *
 *                                                               *
 *                  VERSION 2.0, 08 Sep 1986.                    *
 *      Original version Developed under UNIX on the VAX 11/780  *
 *      by Dr. Ausif Mahmood at Washington State University.     *
 *      Library addition routines, MSDOS, PCDOS and AmigaDOS     *
 *      versions by Scott Kelly at Naval Postgraduate School.    *
 *                                                               *
 *****************************************************************/

int _mlen = 500;
int _mem[500] ;
#include "stdio.h"              /*<stdio.h> in UNIX environment   */

/* This is the parsing and code generation (inter-connection list*/
/* of descriptors for the user circuit) program. It also produces*/
/* the corresponding symbol table for printing outputs. Recursive*/
/* descent  parsing scheme is used.  Syntax directed  translation*/
/* (SDT) scheme is used for code generation.                     */


#define maxkey 29               /* maximum number of keywords      */

#define maxsym 500              /* symbol table size               */
                                /* 1000 size in UNIX               */
#define maxprim 100             /* maximum number of primitives    */

#define maxouts 32              /* maximum of 32 outputs per prim. */

#define maxnorm 50              /* normload table size             */


/*------------------GLOBAL DECLARATIONS-------------------------*/
int strcpy() ;                  /*   copies one string to another */

extern int primsetup();         /*   external module to name the prims */

extern int build();             /*  creates a MULTIMOD file for STRUC */

extern int uexpand();           /*  prints system-generated expansion */
                                /*  requests                          */

extern int struc_expand(); /*  checks to see if extended primitive */
                                /*  descriptions need to be added     */
extern int append();            /*  appends structural descriptions to */
                                /*  the MULTIMOD file if called for   */
extern int countcells();        /*  counts the number of MODULEs in   */
                                /*  the user program                  */
extern int check_deeper(); /*  recursive expansion routine       */

extern int checktable();   /*  checks if primitive in expand table*/
```

```
extern int perform_expansion(); /* routine that performs expands */

int strcmp() ;              /*   string comparison                 */

int getid()  ;              /*   get next identifier               */

int find_token() ;          /*   returns the token number          */
                            /*   (keyword table index)             */
extern int swap();          /*   performs function substitutions */
int COMPILE() ;             /*   parsing routine for 1 MOD.        */
int ADD();                  /*   tests for a request to add cells */
int MOD() ;                 /*   MODULE parsing routine            */
int INP() ;                 /*   INPUTS parsing routine            */
int OUT() ;                 /*   OUTPUTS parsing routine           */
int TYP() ;                 /*   TYPES parsing routine             */
int IDSTRING() ;            /*   list of id's parsing              */
int CKT() ;                 /*   ckt. inter-connection parsing*/
int DEF() ;                 /*   DEFINE parsing routine            */
int INI() ;                 /*   INITIALIZE parsing routine        */
int PRI() ;                 /*   PRINTOUT parsing routine          */
extern int add_lib();       /*   adds primitives to library        */

int rfdel() ;               /*   rise/fall delay handling          */

int bdread() ;              /*   block delay reading routine       */

int cmode() ;               /*   code generation for mode          */

int matgen() ;              /*   delay matrix and mode gen.        */

int parseid() ;             /*   single id parsing                 */

int findid() ;              /*   finds symbol table index          */

int finddesc() ;            /*   finds symbol table index for */
                            /*   the given function name/type */

int updesct() ;             /*   updates the descriptor table */
                            /*   (name and symbol table index)*/

int findprim() ;            /*   finds primitive library index*/

int update() ;              /*   update symbol table               */

int connect() ;             /*   code gen. and fanid update        */
                            /*   (descriptor interconnections)*/
int code_input() ;          /*   code generation for INPUTS        */
                            /*   declaration                       */
int errmessage() ;          /*   error message printing            */

int outerror() ;            /*   output error message              */

int error() ;               /*   error handling routine            */

int firstp() ;              /*   first pass for expand             */
```

```
int secondp() ;              /*   second pass for expand       */

int pdelim() ;               /*   prints a file of keywords and */
                             /*   tokens, separated by delimiters*/

int ftype() ;                /*   finds type of a identifier   */

int reverse() ;              /*   reverses a string            */

int fcopy() ;                /*   file copying routine         */

int copy_noend();            /*   copies everything but END token */

int itoa() ;                 /*   integer to ASCII routine     */

int cexpand() ;              /*   expands the primitive in ckt */

int substitute() ;           /*   substitutes the func's code  */

int foutorder() ;            /*   one to one correspondence for*/
int finorder() ;             /*   actual and formal parameters */

int ckt_line() ;             /*   scans one line of ckt desc.  */

int search() ;               /*   search a delimiter or token  */

int tack() ;                 /*   tacks a number to an id      */

int multi_mod() ;            /*   sub module handling routine  */

int fadvance() ;             /*   advances to next line        */

int hashf() ;                /*   converts input name to number*/

int test();                  /*   tests for hash collisions    */
/*-------------------------------------------------------------*/


/*----------------DATA STRUCTURES--------------------------*/
struct sym_tab {             /*  symbol table                */
  char name[8] ;             /*  name = name of id           */
  int descno, funcno ;       /*  descno = descriptor number  */
  int fanid;                 /*  funcno = primitive lib index*/
 } ;                         /*  fanid = actual circuit load */

struct desc_tab {            /*  table containing function   */
  char fun[8] ;              /*  names (type names) and their*/
  int dnum ;                 /*  symbol table indexes        */
 } ;

struct norm_tab {            /*  table containing function   */
  char nom[8] ;              /*  names/types and associated  */
  int nmld ;                 /*  normload declared in DEFINE */
 } ;

struct prim_tab {            /*  Primitive table :           */
```

```
  char nam[8] ;              /*  primitive table              */
  char nam2[8] ;             /*  EXTENDED primitive table     */
  int numpar, outp ;         /*  numpar = no. of parameters   */
  int normid, fanout ;       /*  for the function             */
  int technology, overid ; /*  outp = # of outputs          */
 } ;

struct err_stack {           /* stack for errors in one line */
  char nm[8] ;               /* nm = name of unexpected id   */
  int errno ;                /* errno = error number         */
 } ;

struct exp_tab {             /* expand table                 */
  char fname[8] ;            /* fnum = prim lib index        */
  int fnum ;
 } ;

struct namepair {            /* this holds system-generated  */
  char e_from[8];            /* expansion requests           */
  char e_to[8];
 };

struct swapname {            /* each of these nodes will contain a module */
  char sname[8];             /* specified in the USING parameter list */
  int used;                  /* this indicates whether used or not    */
  };
/*-------------------------------------------------------*/


/*---------------STORAGE ALLOCATION-----------------------*/
struct sym_tab symt[maxsym] ;
struct desc_tab desct[maxsym] ;
struct norm_tab nort[maxnorm] ;
struct prim_tab prim[maxprim], *primptr ;
struct err_stack errt[5] ; /* max of 5 errors per line     */
struct exp_tab expt[30] ;  /* max 30 expansion requests    */
struct swapname typelist[maxprim][8];  /* table of module replacements*/
struct swapname swaplist[20][8];  /* USING parameter list storage */
struct namepair regtable[maxprim];

int req_count;               /* number of system expand reqs */
int line_count;              /* line index for swaplist      */
int swap_flag;               /* indicates whether all of this*/
                             /* is necessary or not          */

extern int found_start;      /* marks occurrance of SWAPLIN  */
extern int found_end;        /* marks occurrance of ENDSWAP  */

int sfound;

int add_flag;                /* set if cell is to be added to*/
                             /* primitive library            */
int err_ptr ;                /* error table pointer (count)  */
                             /* for one line.                */

int matcount ;               /* delay matrix count           */
```

```
int delimiter, bb, skip ;  /* delimiter = delimiter type    */
                           /* bb = buff[80] index (line)    */

int rdmat[maxouts][maxouts], fdmat[maxouts][maxouts] ;
                           /* rise and fall delay matrices */

int tokno, err_count ;     /* err_count  = error count      */
int desc_no , syn_count, symid ;   /* desc_no = desc. count*/

int dptr, descid, lim ;    /* descriptor table indices      */
                           /* dptr = descriptor table cnt   */

int normcount ;            /* normtable count               */

int expcount ;             /* expand table count            */
int cellcount;             /* # of MODULEs in user program */
int filecount,pct;         /* poutcount used for debugging */
int prinoflag;             /* controls printing of source  */
int print_select;          /* determines whether a ')' causes*/
                           /* a linefeed or not (default=no)*/

int prim_count, primid ;   /* prim_count = # of primitives */
int sys_prims ;            /* number of permanent (system) */
                           /* primitives                   */
int savprim ;
int inpcount,outpcount;    /* number of inputs and outputs */
                           /*(used to add a new primitive) */

int features[maxprim][2];  /* first field describes the type of */
                           /* descriptions available;           */
                           /* -1 => empty     0 => block only   */
                           /* 1 => struc only  2 =>block & struc */
                           /* second field indicates primitive level */

int append_table[maxprim]; /* keeps track of the functions we've */
int append_index;          /* added to the user program         */
int expdone;               /* marks completion of struc expansion*/
int no_compilation;        /* used when only making library additions*/
char token_buf[8], savbuf[8], buff[80] ; /* buff = 1 line  */
char keyword[maxkey][8] ;  /* keyword table                 */
char inch;
char instack[maxouts][8], outstack[maxouts][8] ;
char in1stack[maxouts][8], out1stack[maxouts][8] ;
char in2stack[maxouts][8], out2stack[maxouts][8] ;
/* in/out stack = inouts/outputs from primitive's definition in  */
/* library. in1/out1 = calling inouts/outputs (user program)     */
/* in2/out2 = inputs/outputs of each line in primitive's desc.   */

char typstack[maxouts][8] ;/* types of primitive to be expanded  */
int typcount ;

int hashtable[100];             /* the hash table                */
int hashcount;                  /* number of items in hashtable  */

int incount , outcount ;
int in1count, out1count, in2count, out2count ;
```

```
int outorder, inorder ;
int count ;             /* for orinting on the file */
char savfunc[8], userprg[8] ;
int ft, occurance ;     /* occurance = # of times call to    */
                        /* primitive to be expanded is made  */

FILE *r1 ;              /* oointer to inout data file        */
FILE *r2 ;              /* oointer to STRUCT library         */
FILE *r3 ;              /* oointer to modified STRUC library */
FILE *r4 ;              /* pointer to user STRUC description */
FILE *w1 ;              /* pointer to expanded file   P1EXP  */
FILE *t1 ;              /* oointer to temp file              */
FILE *lii ;             /* pointer to library                */
FILE *s1 ;              /* oointer to primitive's desc, SCR1 */
FILE *s2 ;              /* pointer to expanded circuit  SCR2 */
FILE *rp ;              /* read oointer to input data file   */
FILE *wq ;              /* circuit description for simulator */
/*-------------------------------------------------------------*/


/*----------------------MAIN PROGRAM---------------------------*/
main(argc, argv)
int argc ;
char *argv[];
 {
  int i, j;

  strcpy(userprg,argv[1]);
  prinpflag = 0 ;
  filecount = 0 ;
  inpcount=0;
  outpcount=0;
  no_compilation=0;              /* always assume we're compiling */
  print_select=0;                /* print ')' without a linefeed  */
  req_count=0;
  swap_flag=0;
  line_count=0;
  hashcount=0;
  for (i=1; i<100; i++)
     hashtable[i] = -1;

/*--------------------PRIMITIVES SUPPORTED--------------------*/
  for (i = 0; i < maxprim; i = i + 1)
   {
    primt[i].normld = 1 ;
    primt[i].fanout = 20 ;
    primt[i].technology = 0 ;
    primt[i].overld = 5 ;
   }

   primsetup(&primt[0]);             /* initialize primitives */
   sys_prims=prim_count;             /* primcount may change, but */
                                     /* we need a copy of its     */
                                     /* starting value            */

/*------------------INITIALIZE-------------------------------*/
```

```
 for (i = 0; i < maxsym; i = i + 1)
  {
   symt[i].fanid = 0 ;
   symt[i].descno = -1 ;
   symt[i].funcno = -1 ;
  }
 for (i = 0; i < 20 ; i = i + 1)
  exot[i].fnum = -1 ;

 for (i=0; i<20; i++)
    {
    for (j=0; j<8; j++)
       {
       swaolist[i][j].used=0;          /* set USING parameter lists */
       }                               /* to EMPTY                  */
    }

 for (i=0; i<maxprim; i++)
    {
    for (j=0; j<8; j++)
       {
       typelist[i][j].used=0;          /* set type list to empty */
       }
    }

/*-------------------------------------------------------------*/

/*----------------------KEYWORDS-------------------------------*/
 strcpy(keyword[0], "MODULE") ;
 strcpy(keyword[1], "INPUTS") ;
 strcpy(keyword[2], "OUTPUTS") ;
 strcpy(keyword[3], "TYPES") ;
 strcpy(keyword[4], "{") ;
 strcpy(keyword[5], "}") ;
 strcpy(keyword[6], "INITIAL") ;
 strcpy(keyword[7], "PRINTOU") ;
 strcpy(keyword[8], "INTERNA") ;
 strcpy(keyword[9], "DEFINE") ;
 strcpy(keyword[10],"RISEDEL") ;
 strcpy(keyword[11],"FALLDEL") ;
 strcpy(keyword[12],"TECHNOL") ;
 strcpy(keyword[13],"TTL") ;
 strcpy(keyword[14],"NMOS") ;
 strcpy(keyword[15],"CMOS") ;
 strcpy(keyword[16],"ECL") ;
 strcpy(keyword[17],"FANOUT") ;
 strcpy(keyword[18],"NORMLOA") ;
 strcpy(keyword[19],"OVERLOA") ;
 strcpy(keyword[20],"END") ;
 strcpy(keyword[21],"EXPAND") ;
 strcpy(keyword[22],"USING");              /* used to replace modules*/
 strcpy(keyword[23],"SWAPLIN");            /* marks each ckt line  */
 strcpy(keyword[24],"ENDSWAP");            /* to be examined for swaps*/
 strcpy(keyword[25],"NOEXP");
 strcpy(keyword[26],"ADDLIB");             /* add a cell primitive */
 strcpy(keyword[27],"ADSTRUC");            /* struc-only description*/
```

```
 strcpy(keyword[28],"ADBLOCK");              /* block-only description*/
/*-------------------------------------------------------------*/

  err_ptr = -1 ;          /* error count for one line   */
  err_count = 0 ;               /* error count for program    */
  expcount = 0 ;
  printf("Opening the circuit descriptor file....\n");

  for (i=0; i<maxprim; i++)               /* initialize the append table */
     append_table[i] = -1;                /* to empty                    */


  append_index=0;
  cellcount=0;
  r1=fopen(argv[1],"r");
  countcells(r1);                         /* count the # of MODULEs  */
  fclose(r1);

  r1=fopen(argv[1],"r");
  r2=fopen("outfile","w");            /* the "stripped" file       */

  build();                                /* strip off the END keyword */
  fprintf(r2," END; \n");
  fclose(r2);

  if ((no_compilation==1) && (add_flag==1))  /* no_compilation set? */
     {
     add_lib();                  /* yes, add the modules without compiling*/
     }
  else
     {                                    /* no, begin compilation */

     r1=fopen("outfile","r");
     w1=fopen("p11","w");               /* copy user prog to "p11"  */
     fcopy(r1,w1);                        /* (now we're back to       */
     fclose(r1);                          /* Ausif's code)            */
     fclose(w1);
     printf("Files are restored.  Multimodule expansion begins.\n");
     /*-----------------------EXPANSION-------------------------------*/
     firstp() ;   /* first pass, determine any expansion requests,  */
                  /* put each request on expand table (expt)        */
     perform_expansion();      /* any expansions handled in here    */

     r1=fopen("p11","r");          /* copy p11 to INFILE */
     r2=fopen("infile","w");
     fcopy(r1,r2);
     fclose(r1);
     fprintf(r2, "END; \n");
     fclose(r2);

     r1=fopen("infile","r");        /* copy INFILE to OUTFILE */
     r2=fopen("outfile","w");       /* but leave OUTFILE open */
     copy_noend(r1,r2);
     fclose(r1);

     cellcount=0;
```

```
    r1=fopen("infile","r");
    countcells(r1);                    /* count the # of MODULEs  */
    fclose(r1);
    r1=fopen("infile","r");
    struc_expand();                    /* handle any struc-only prims */

    r1=fopen("outfile","r");          /* copy this file back to p11 */
    r2=fopen("p11","w");              /* and expand the struc-only prims */
    fcopy(r1,r2);
    fclose(r1);
    fclose(r2);
    for (i=0; i<expcount; i++)
       expt[i].fnum = -1;              /* clear the expand table */
    expcount=0;

    firsto();                          /* and take care of any modules */
    if (expcount>0)                    /* that struc_expand() added    */
       perform_expansion();

    if (swap_flag==1)                  /* any USINGS to deal with? */
       swap();                         /* if so, make the substitutions */

    /*------------------end expansion-----------------------------*/
    rp = fopen("p11","r");             /* expanded user program */
    wq = fopen("simdata","w");
                                       /* initialize compiler vars */
    orinpflag = 1 ;

    dptr = 0 ;                         /* desctable count              */
    normcount = 0 ;                    /* norm table count             */
    sym_count = 0 ;                    /* symble table entries count */
    desc_no = 0 ;                      /* descriptor count             */
    symid = -1 ;                       /* symbol table index           */
    matcount = 0 ;                     /* delay matrix count           */

    /*-----------------PARSING AND CODE GENERATION-------------------*/
    /* Recursive descent parsing is used . STD scheme is used for    */
    /* code generation. BNF is as follows.                           */
    /*                                                               */
    /* <COMPILE> => <MOD> <INP> <OUT> <TYP> { <CKT> } <DEF> <INI> <PRI>*/
    /* Non-terminals are defined in their respective sub-programs    */

    COMPILE() ;

    fclose (rp) ;
    if (err_count != 0)
       error(26) ;                /* Compilation discontinued message */
    else
       error(38) ;                /* no errors encountered message    */
    outerror() ;
    }

}
/*-----------------END OF MAIN PROGRAM---------------------------*/
```

```
/*---------------MAIN PARSING ROUTINE FOR 1 MODULE---------------*/
/* Recursive descent parser.  Synatx directed translation (SDT)  */
/* scheme is used for code generation.  BNF is as follows        */
/*                                                               */
/* <COMPILE> => <MOD> <INP> <OUT> <TYP> { <CKT> } <DEF> <INI> <PRI>*/
/* <--> = non terminals, all others are terminals                */

 COMPILE()
  {
   printf("Compilation begins.\n");
   MOD() ;              /* call to MODULE parsing routine   */
   INP() ;              /* call to INPUTS parsing routine   */
   OUT() ;              /* call to OUTPUTS parsing routine  */
   skip = 0 ;
   TYP() ;              /* call to TYPES parsing routine    */

   if (skip != 1)
    parseid(4) ;        /* '{' parsing                      */
   skip = 0 ;
   CKT() ;              /* Circuit interconnections parsing */
                        /* '}' taken care of in CKT         */
   matgen() ;           /* delay matrix generator           */
   DEF() ;              /* DEFINE parsing                   */
   INI() ;              /* INITIALIZE parsing               */
   PRI() ;              /* PRINTOUT parsing                 */
   fprintf(wq," 50\n") ;  /* put a terminator on "simdata" */
   fclose(wq);          /* clean up and quit                */
   if (add_flag==1)
      {
      add_lib();         /* perform additions to primitive library */
      }
  }
/*---------------------------------------------------------------*/


/*-----------------MOD()-----------------------------------------*/
/* <MOD> => MODULE <delimiter> id                                */
 MOD()
  {
   parseid(0) ;                 /* ADD and MODULE parsing  */
   parseid(maxkey) ;                /* module name             */
  }
/*---------------------------------------------------------------*/


/*-----------------INP()-----------------------------------------*/
/* <INP> => INPUTS <delimiter> <IDSTRING>                        */
 INP()
  {
   parseid(1) ;                     /* INPUTS parsing          */
   IDSTRING(0) ;                    /* input names             */
  }
/*---------------------------------------------------------------*/


/*-----------------OUT()-----------------------------------------*/
```

```
/* <OUT> => OUTPUTS <delimiter> <IDSTRING>                          */
 OUT()
 {
  parseid(2) ;              /* OUTPUTS parsing                     */
  IDSTRING(-1) ;            /* -1 = outputs code for sym tab       */
 }
/*-----------------------------------------------------------------*/


/*--------------------TYP()----------------------------------------*/
/* <TYP> => TYPES <delimiter> <T>                                  */
/* <T> => <PRIMTYPE> | <INT TYPE> | ;                              */
/* <PRIMTYPE> => <PRIMITIVE> = <IDSTRING>                          */
/* <INT TYPE> => INTERNALS = <IDSTRING>                            */

/* '(' parsing is covered in this routine                          */

 TYP()
 {
  parseid(3) ;                      /* TYPES parsing               */

  while (1)                         /* <T> expansion               */
   {
    getid(rp,41) ;            /* 41 = missing ( error    */
    find_token() ;
    if (token == 4)           /* if '(' found, then quit */
     {
      skip = 1 ;
      break ;                 /* no types declared       */
     }
    if (token == 8)           /* <INT. TYPE> expansion   */
      IDSTRING(-2)  ;         /* -2 = code for internals */
    else      /* TYPES expansion    */
     {
      findprim() ;            /* <PRIMTYPE> expansion    */
      if (primid >= prim_count)
        error(30) ;           /* undefined fuction       */

      IDSTRING(primid) ;
     }     /* end TYPES   */
   }       /* end while 1 */
 }         /* end function*/
/*-----------------------------------------------------------------*/


/*--------------------IDSTRING()-----------------------------------*/
/* <IDSTRING> => id ; | <IDSTRING> id ,                            */
 IDSTRING(code)          /* code = 0 for inputs,-1 for outputs     */
 {                       /* -2 for internals,                      */

    while (delimiter != 2)          /* delimiter = ;       */
     {
      parseid(maxkey) ;             /* name                */
      update(code) ;                /* update symbol table */
                                    /* code = 0 for input  */
                                    /* -1 for output       */
```

```
                                    /* prim # for TYPE      */
      if (code == 0 )
        code_input(desc_no) ;       /* input code gen.      */

      if (code <= 0 )
        desc_no = desc_no + 1 ;

     }
   }
/*-------------------------------------------------------------*/


/*--------<CKT>---INTERCONNECTIONS parsing----------------------*/
/* <CKT> => <IDSERIES> = <PRIM> ( <IDSERIES> ) ; <CKT> |       */
/*                       <IDSERIES> = <PRIM> ( <IDSERIES> ); */
/* <IDSERIES> => id | id, <IDSERIES>                           */

 CKT()
 {
  int outpar, end ;         /* number of output parameters     */
  int savoar, j ;
  int savid[maxouts], savn[maxouts], fwdp ;
  /* savid[] saves symbol table indexes while savn[] saves desc. */
  /* numbers for output list names                             */

  end = 0 ;
  while (end == 0)
   {
/*-----------------------<IDSERIES> for outputs----------------*/
    outpar = -1 ;
    while (1)
     {
      getid(rp,46) ;        /* left side of assignment statement   */
      find_token() ;

      if (token == 5)       /*  if } found, end compilation        */
       {
        end = 1 ;
        break ;
       }
      if (token < maxkey) /* left hand side should not be a keyw.*/
        error(25) ;
      outpar = outpar + 1 ;

      findprim() ;
      if (primid < prim_count)
        error(25) ;         /* output name is a keyword            */

      findid() ;            /* find the symbol table index         */
      if (symid >= 0)
       {                        /* save output indices in an array */
        savid[outpar] = symid ;
        savn[outpar] = synt[symid].descno ;
       }

      if (delimiter == 4) /* '=' should follow the output names  */
```

```
        break ;
      else
       {
        if (delimiter != 1)
          error(31) ;      /* ',' expected after each name        */
       }
     }                     /* end while <IDSERIES> for outputs    */
/*-------------------------------------------------------------*/
    if (end == 1)     /* if ')' found quit */
      break ;

/*--------------------------<PRIM>-------------------------------*/
    getid(rp,33) ;         /*  function name                     */
    if (delimiter != 6)    /*  '(' should follow function name   */
      error(29) ;

    if (err_count == 0)
     {
      if (outpar > 0)      /* if # of outputs > 1, connect exten- */
       {                   /* sion pointers.                      */
        for (j = 0; j < outpar; j = j + 1)
         {
          /* fprintf(wp, " desc[%d].ext_ptr = &(desc[%d]) ;\n",  */
          fprintf(wq, " 1 %d 15 %d",
                                      savn[j], savn[j+1]) ;
          /*  fprintf(wp," desc[%d].parent = %d ;\n",            */
          fprintf(wq," 1 %d 16 %d",
                                      savn[j+1], savn[0]) ;
          fadvance(8) ;
         }
       }
     }
    findprim() ;
    if (primid >= prim_count)
     {
      findid() ;              /* function name is a type */
      primid = symt[symid].funcno ;
     }

    if (err_count == 0)
    /* fprintf(wp," desc[%d].pfunc=%s;\n",savn[0],      */
    /*                            prmt[primid].nam) ; */
       fprintf(wq," 33 %d %d",savn[0],primid) ;
    fadvance(3) ;

    if (prmt[primid].outp != (outpar+1))
      error(35) ;          /* # of outputs should be as in table  */

    for(j = 0; j <= outpar; j = j + 1)   /* update symbol table  */
     {                                   /* and desc table       */
      symt[(savid[j])].funcno = primid ;
      updesct(token_buf, savid[j]) ;
     }
/*-------------------------------------------------------------*/
```

```
/*--------------------<IDSEPTES> for inputs----------------------*/
    fwdp = 0 ;
    savpar = primt[primid].numpar ;  /* number of inputs          */
    savprim = primid ;
    strcpy(savbuf, token_buf) ;  /* save function name/type       */

    while (savpar != 0)    /* while all inputs have been scanned */
     {
      parseid(maxkey) ;    /* parameter of function             */
      findid() ;           /* find parameter's location in the  */

      connect(0,savn,fwdp);/* generate code and update fanid    */

      if (savpar == 1)
       {
        if (delimiter != 5)  /* ')' expected after the last arg. */
         error(32) ;
       }
      savpar = savpar - 1 ;
      if (savpar != 0)
       {
        if (delimiter != 1)  /*, expected after first parameter  */
          error(31) ;

        parseid(maxkey) ;    /*  get next argument               */
        if (savpar > 1)
         {
          if (delimiter != 1)/* , expected after argument        */
            error(31) ;
         }
        else
         {
          if (delimiter != 5)
            error(32) ;       /* ) expected after last arg.     */
         }
        findid() ;            /* find symbol table index for the */
                              /* input name                      */
        connect(1,savn,fwdp); /* generate code and update fanid  */

        savpar = savpar - 1 ;
        fwdp = fwdp + 1 ;

        if (outpar > 0)             /* multioutput case */
          outpar = outpar - 1 ;
        else
         {
          if (savpar != 0)          /* multiinput case  */
           {
        /*  fprintf(wp," desc[%d].ext_ptr = &(desc[%d]);\n",     */
            fprintf(wq," 1 %d 15 %d",
                                savn[fwdp - 1], desc_no) ;
        /*  fprintf(wp," desc[%d].parent = %d ;\n",              */
            fprintf(wq," 1 %d 16 %d",
                                 desc_no, savn[fwdp-1]);
            fadvance(8) ;
```

```
            savn[fwdo] = desc_no ;
            desc_no = desc_no + 1 ;
           }
         }
       } /* end if                          */
     }   /* end of <IDSERIES> for inputs */
   }     /* end while end = 0               */
 }       /* end <CKT>                       */
/*-------------------------------------------------------------*/


/*---------------------CONNECT---------------------------------*/
/* Circular list generation for the circuit. Previous list is  */
/* broken and new circular looo is made.                       */

 connect(f,savn,fwdo)
 int f ;     /* f is 0 or 1 */
 int savn[], fwdp ;      /* savn has desc # of output names */
 {
  int i ;

  if (err_count == 0)
   {
    /*-------------------------update fanid-------------------------*/
    for (i = 0; i < normcount; i = i + 1)  /* find name in nort  */
     if (strcmo(nort[i].nom,savbuf) == 0)
       break;
    if (i < normcount)     /* if over ride is used for norm load */
      symt[symid].fanid = symt[symid].fanid + nort[i].nmid ;
    else                   /* use default value from prim. lib   */
      symt[symid].fanid=symt[symid].fanid + primt[savorim].normid;
    /*-------------------------------------------------------------*/

/*  fprintf(wo," savn = desc[%d].h_value ;\n",        */
    fprintf(wq," 2 %d",
                                 symt[symid].descno) ;
/*  fprintf(wp," otr = desc[%d].header ; \n",         */
    fprintf(wq," 3 %d",
                                 symt[symid].descno) ;
                      /* save current oointer from the input   */

/*  fprintf(wo,"desc[%d].header=&(desc[%d]);\n",      */
    fprintf(wq," 1 %d 8 %d",
                symt[symid].descno, savn[fwdp]) ;
/*  fprintf(wp,"desc[%d].h_value = %d ;\n",symt[symid].descno,f);*/
    fprintf(wq," 1 %d 11 %d",symt[symid].descno,f);
/*  fprintf(wo,"desc[%d].right%d = ptr ;\n",   */
    fprintf(wq," 1 %d 9 %d",
                                        savn[fwdo],f) ;
/*  fprintf(wo,"desc[%d].r_value[%d] = savh ;\n",    */
    fprintf(wq," 1 %d 12 %d",
                                        savn[fwdp], f) ;
                             /* complete the C-list             */
    fprintf(wq," \n");
    filecount = 0 ;
   }
```

```
 }     /* end connect  */
/*-------------------------------------------------------------*/


/*-------------------------DEFINE parsing----------------------*/
/* <DEF> => <PRIMID> : <DEFINITIONS>                           */
/* <DEFINITIONS> => RISEDELAY(num, num) = num  |               */
/*                  FALLDELAY(num, num) = num  |               */
/*                  FANOUT = num               |               */
/*                  NORMLOAD = num             |               */
/*                  OVERLOAD = num             |               */
/*                  TECHNOLOGY = <TECHTYPE>                    */
/* <TECH TYPE> = TTL | NMOS | CMOS | ECL                       */

DEF()
 {
  int j, num, savtoken ;
  int fan1, tech1, over1 ;

  skip = 0 ;
  parseid(9) ;       /* DEFINE expected */
  while (1)
   {
    getid(rp,42) ;
    find_token();              /* function name or type          */

    if (tokno == 6)            /*   if token = 'INITIALIZE'      */
     {
      skip = 1;
      break ;
     }

    strcpy(savbuf, token_buf);/*  save function name in savbuf   */
    findprim() ;
    if ( primid >= prim_count)
     {
      findid () ;
      primid = symt[symid].funcno ;
     }

    fan1  = primt[primid].fanout ;
    tech1 = primt[primid].technology ; /* default parameters     */
    over1 = primt[primid].overld ;

/*-------------------------DEFINE statement--------------------*/
    while (delimiter != 2)     /*  each DEFINE ends with ';'     */
     {
      getid(rp,43) ;
      find_token();
      savtoken = tokno ;       /*  savtoken = 'RISEDELAY' ...etc..*/

      switch(savtoken) {
        case 10: rfdel(0) ;               /* rise delay        */
                 break ;

        case 11: rfdel(1) ;               /* fall delay        */
```

```
                break ;

        case 12: getid(rp,33) ;           /* TECHNOLOGY = --   */
                 for (j = 13; j <= 16; j = j + 1)
                  {
                   if (strcmp(token_buf, keyword[j]) == 0)
                     break ;
                  }
                 if (j > 16)
                   error(36) ;            /* undefined technol.*/
                 else
                   tech1 = j ;
                 break ;

        case 17: getid(rp,33);/* FANOUT    =  get value of para.*/
                 fan1 = atoi(token_buf) ; /* ASCII to integer  */
                 break ;

        case 18: break ;     /* normload handled in first pass */

        case 19: getid(rp,33) ;
                 over1 = atoi(token_buf) ;/* value of overld   */
                 break ;

        default: error(36) ;  /* syntax error message */
       }     /* end switch  */
     }       /* while ;  each DEFINE ends with ';' */

    /*--------------------generate code for mode-----------------*/
    if (err_count == 0)
     {
      lim = 0;
      while (lim < dptr)
       {
        finddesc(savbuf) ;
        if (lim > dptr)
          break ;
        num = symt[descid].fanld ;
        if (num > fan1)
          cmode(num, fan1, over1, tech1) ;
       }
     }
    /*-----------------------------------------------------------*/
   }       /* end while (1) */
 }         /* end DEFINE    */
/*-------------------------------------------------------------*/


/*----------------------CMODE()--------------------------------*/
/* Code generator for mode. Code is generated only if mode != 0  */

 cmode(num, fan1, over1, tech1)
 int num, fan1, over1, tech1 ;
 {
  if (num <= (fan1 + over1))
   {
```

```
    if (tech1 == 16)
  /*  fprintf(wp," desc[%d].param[6] = 2 ;\n",    */
      fprintf(wq," 1 %d 6 2",
                                        symt[descid].descno) ;
    else
  /*  fprintf(wp," desc[%d].param[6] = 1 ;\n",    */
      fprintf(wq," 1 %d 6 1",
                                        symt[descid].descno) ;
   }
  else
   {
    if (tech1 == 16)
  /*  fprintf(wp," desc[%d].param[6] = 3; \n",    */
      fprintf(wq," 1 %d 6 3",
                                        symt[descid].descno) ;
    else
  /*  fprintf(wp," desc[%d].param[6] = 4; \n",    */
      fprintf(wq," 1 %d 6 4",
                                        symt[descid].descno) ;
    fadvance(4) ;
   }
 }   /* end CMODE */
/*-------------------------------------------------------------*/


/*--------------------RFDEL() ---------------------------------*/
 rfdel(n1)
 int n1 ;
 {
  int par1, par2, parm1, parm2, num, savpar1, savpar2, inp ;

  if (delimiter != 6)
    error(29) ;          /* '(' expected after RD, FD */
  getid(rp,33) ;
  par1 = atoi(token_buf) ;             /* first index */

  parm1 = prim[primid].numpar ;
  parm2 = prim[primid].outp ;
  if (par1 > parm1)
    error(39) ;          /* incorrect first index  */

  if (delimiter != 1)       /* ',' expected           */
    error(31) ;

  getid(rp,33) ;
  par2 = atoi(token_buf) ;            /* second index */
  if (par2 > parm2)
    error(40) ;
  if (delimiter != 5)
    error(32) ;          /* ')' expected         */

  getid(rp,33) ;         /* value of rise delay  */
  num = atoi(token_buf) ;

  savpar1 = par1 ;         /* save first index     */
  savpar2 = par2 ;         /* save second index    */
```

```
  lim = 0 ;
   printf("dptr=%d\n",dptr);
  while ( lim < dptr)      /* generate code for all*/
   {                       /* descs. using name in */
    finddesc(savbuf) ;     /* savbuf               */

    lim = lim + parm2 - 1 ; /* lim is incremented by # of outputs*/
    printf("new val for lim is %d\n",lim);
    if (lim > dptr)          /* desctable has successive entries  */
      break ;                /* for multi-output descriptors      */

    fadvance(2) ;

    if (pari > 1)          /* first access desc.   */
     {
/*    fprintf(wp," ptr = desc[%d].ext_ptr;\n",symt[descid].descno);*/
      fprintf(wq," 6 %d",symt[descid].descno);
      pari = pari - 2 ;
      while( pari > 0)
       {
  /*    fprintf(wp," ptr = (*ptr).ext_ptr ;\n") ;    */
        fprintf(wq," 7 ");

       fadvance(1) ;

        pari = pari - 2 ;
       }
     }    /* if pari > 1 */
    else                   /* if pari = 1          */
/*   fprintf(wp," ptr = &(desc[%d]);\n",symt[descid].descno);*/
     fprintf(wq," 4 %d",symt[descid].descno);

    inp = savpari % 2 ;    /* even or odd pari     */
    if (par2 == 0)
/*    fprintf(wp," (*ptr).param[%d] = %d ;\n",     */
      fprintf(wq," 5 %d %d",
                                ((2+ni) + 2*inp), num) ;
    else                   /* multi-output case    */
     {
      par2 = par2 - 1 ;
/*    fprintf(wp," ptrm = (*ptr).mptr ;\n") ;      */
      fprintf(wq," 8 ") ;

      fadvance(3) ;

      while (par2 > 0)
       {
    /*  fprintf(wp," ptrm = (*ptrm).matptr; \n");  */
        fprintf(wq," 9 ") ;

        fadvance(1) ;

        par2 = par2 - 1 ;
       }
      if (inp  == 0)
       {
```

```
        if (n1 == 0)
       /* fprintf(wp," (*ptrm).rdelay0 = %d;\n",num);   */
          fprintf(wq," 10 %d",num) ;
        else
       /* fprintf(wp," (*ptrm).fdelay0 = %d;\n",num);   */
          fprintf(wq," 11 %d",num) ;
       }
      else
       {
        if (n1 == 0)
/*       fprintf(wp," (*ptrm).rdelay1 = %d;\n",num);    */
          fprintf(wq," 12 %d",num) ;
        else
/*       fprintf(wp," (*ptrm).fdelay1 = %d;\n",num);    */
          fprintf(wq," 13 %d",num) ;

        fadvance(2) ;
       }
     }
    par1 = savpar1 ;
    par2 = savpar2 ;
   }
 }      /* end RFDEL */
/*-------------------------------------------------------------*/


/*-------------------DELAY MATRIX GENERATOR--------------------*/
/* Also generates default mode values for all functions involved */

matgen()
{
 int par1, par2, i, j, k, l, m ;
 int num, f1, o1, t1 ;
 char s[8];

 /*----------------------DEFAULT MODE GENERATION---------------*/
 for (i = 0; i < sym_count; i = i + 1)
  {
   if (symt[i].descno >= 0)
    {
     if (symt[i].funcno > 0)
      {
       descid = i ;      /* cmode() needs descid for code gen.   */
       num = symt[i].fanid ;
       f1 = prmt[(symt[i].funcno)].fanout ;
       o1 = prmt[(symt[i].funcno)].overld ;
       t1 = prmt[(symt[i].funcno)].technology ;
       if (num > f1)     /* if non zero mode */
        cmode(num, f1, o1, t1) ;
      }
    }
  }
 /*-------------------------------------------------------------*/

 /*-------------------DEFAULT DELAYS AND MATRIX STRUCTURE-------*/
 i = 0 ;
```

```
while( i < dptr)
 {
  j = symt[desct[i].dnum].descno  ;      /* descriptor number */
  k = symt[j].funcno ;                   /* function number   */

  strcpy(s, primt[k].nam2) ;   /* s contains name of function */

  bdread(s) ;      /* read block delays for s in rd/fdmat     */

  if (k >= 0)       /* if not input or types etc.*/
   {
    par1 = primt[k].numpar ;
    par2 = primt[k].puto   ;
    i =  i + par2 - 1 ;
         /* par1 = number of inputs, par2 = number of outputs  */

    for (l = 1; l <= par1; l = l + 2)   /* vertical scanning   */
     {
      if (l >  2)
       {
        if (l > 4)
  /*      fprintf(wo," ptr = (*ptr).ext_ptr ;\n") ;     */
          fprintf(wq," 7 ");
        else
  /*      fprintf(wo," ptr = desc[%d].ext_ptr ; \n",j) ;  */
          fprintf(wq," 6 %d",j) ;
       }
      else            /* inputs = 1 or 2 */
  /*    fprintf(wo," ptr = &(desc[%d]) ; \n",j) ;       */
        fprintf(wq," 4 %d",j) ;

      fadvance(2) ;
      /*------------------code for block delays----------------*/
      if (rdmat[l-1][0] != -1)
  /*    fprintf(wo,"(*ptr).param[2] = %d;\n",rdmat[l-1][0]) ; */
        fprintf(wq," 5 2 %d",rdmat[l-1][0]) ;
      if (fdmat[l-1][0] != -1)
  /*    fprintf(wo," (*ptr).param[3] = %d;\n",fdmat[l-1][0]); */
        fprintf(wq," 5 3 %d",fdmat[l-1][0]) ;
      if ((l+1) <= par1)
       {
        if (rdmat[l][0] != -1)
  /*      fprintf(wo," (*ptr).param[4] = %d;\n",rdmat[l][0]); */
          fprintf(wq," 5 4 %d",rdmat[l][0]) ;
        if (fdmat[l][0] != -1)
    /*    fprintf(wo," (*ptr).param[5] = %d;\n",fdmat[l][0]); */
          fprintf(wq," 5 5 %d",fdmat[l][0]) ;
       }
      fadvance(12) ;
      /*-----------------------------------------------------*/
      for (m = 2; m <= par2 ; m = m + 1)
       {
        if (m == 2)
         {
     /*   fprintf(wo," (*ptr).matr = &(matx[%d]);\n",matcount);*/
          fprintf(wq," 14 %d",matcount) ;
```

```
            matcount = matcount + 1 ;
           }
          else       /* if number of outputs > 2 */
           {
        /*  fprintf(wo," matx[%d].matptr = &(matx[%d]) ;\n", */
            fprintf(wq," 15 %d %d",
                                    matcount - 1, matcount) ;
            matcount = matcount + 1 ;
           }
          fadvance(3) ;
          /*-----------------code for block delays--------------*/
          if (rdmat[l-1][m-1] != -1)
        /*   fprintf(wo," matx[%d].rdelay0= %d;\n",matcount-1, */
             fprintf(wq," 16 %d %d",matcount-1,
                                          rdmat[l-1][m-1]);
          if (fdmat[l-1][m-1] != -1)
        /*   fprintf(wo," matx[%d].fdelay0= %d;\n",matcount-1, */
             fprintf(wq," 17 %d %d",matcount-1,
                                          fdmat[l-1][m-1]) ;
          if ((l+1) <= oari)
           {
            if (rdmat[l][m-1] != -1)
         /*   fprintf(wo," matx[%d].rdelay1= %d;\n",matcount-1,*/
             fprintf(wq," 18 %d %d",matcount-1,
                                           rdmat[l][m-1]) ;
            if (fdmat[l][m-1] != -1)
         /*   fprintf(wo," matx[%d].fdelay1= %d;\n",matcount-1,*/
             fprintf(wq," 19 %d %d",matcount-1,
                                           fdmat[l][m-1]) ;
           }
          fadvance(12) ;
          /*-----------------------------------------------------*/
        }   /* end for m = -- */
       }    /* end for l = -- */
      }     /* end if k >= 0  */
     i = i + 1 ;
    }      /* end for i = -- */
 }  /* end matgen */
/*-------------------------------------------------------------*/


/*-------------------BDREAD()----------------------------------*/
/* This routine reads the block delays for a given function name */
 bdread(s)
 char s[8] ;
 {
  int i, j, num1, x, y, w, z, ii ;
  FILE *r1 ;
  char b[8] ;

  r1 = fopen("bldel","r") ;      /* block delay file */

/*-----------------initialize delay matrix to -1 -------------*/
  for (i = 0; i < maxouts; i = i + 1)
   {
    for (j = 0; j < maxouts; j = j + 1)
```

```
      {
       rdmat[i][j] = -1 ;
       fdmat[i][j] = -1 ;
      }
    }
/*-------------------read default block delays-----------------*/
  fscanf(ri,"%s",p) ;
  while (strcmp(p,"END") != 0)
   {
    fscanf(ri,"%d",&num1) ;
    for (i1 = 1; i1 <= num1; i1 = i1 + 1)
     {
      fscanf(ri,"%d %d %d %d",&x, &y, &w, &z) ;
      if (strcmp(p,s) == 0)
       {
        rdmat[x][y] = w ;
        fdmat[x][y] = z ;
       }
     }
    if (strcmp(p,s) == 0)
      break ;
    fscanf(ri,"%s",p) ;
   }
  fclose(ri) ;
 }
/*-----------------------------------------------------------*/


/*--------------------INITIALIZE parsing-------------------------*/
/* <INI> => INITIALIZE ; <INITSTRING>                            */
/* <INITSTRING> => id = num , <INITSTRING> | id = num ;          */
 INI()
 {
  int flag ;

  flag = 0 ;                    /* begint[0] = NULL indicator     */
  if (skip != 1)
   parseid(6) ;
  while (delimiter != 2)        /*  INITIALIZE ends with ';'      */
   {
    parseid(maxkey) ;
    if (delimiter != 4)         /* '=' expected after the name     */
      error(27) ;
    findid() ;                  /* symbol table index              */

    getid(rp,43).;              /*  initialization value           */
    if (err_count == 0)
     {
   /* fprintf(wp,"depth[0] = depth[0] + 1 ;\n") ;  */
      fprintf(wq," 20") ;
      if (flag == 0)
        fprintf(wq," 21") ;
   /*   fprintf(wp," begint[0] = freet ; \n") ;    */
      else
        fprintf(wq," 22") ;
   /*   fprintf(wp," (*(endt[0])).tptr = freet ;\n" );  */
```

```
      /* allocate storage for tstack */
    /* fprintf(wp," endt[0] = freet ; \n");          */
    /* fprintf(wo," freet = (*freet).tptr ;\n") ;    */
    /* fprintf(wp," (*(endt[0])).tptr = NULL ; \n") ;*/
       fprintf(wq," 23 24 25") ;

       /* generate code */
    /* fprintf(wp," (*(endt[0])).dptr = &(desc[%d]) ;\n",       */
       fprintf(wq," 26 %d",
                                        symt[symid].descno) ;
    /* fprintf(wp," (*(endt[0])).newval = %s ;\n",token_buf) ;  */
       fprintf(wq," 27 %s",token_buf) ;
       flag = 1 ;
       fadvance(6) ;
      }
    }
  }
/*-------------------------------------------------------------*/


/*--------------------'PRINTOUT' parsing-----------------------*/
/* <PRI> => PRINTOUT : <IDSTRING>                              */
PRI()
{
 int i, j ;

 parseid(7) ;
 i = 0 ;
 while (delimiter != 2)        /* PRINTOUT ends with ';'       */
  {
   parseid(maxkey) ;
   findid() ;
   if (err_count == 0)
    {
     for (j = 0; j <= 7 ; j = j + 1 )
      {
       if (token_buf[j] == '\0')
         break ;
       else
        {
    /*   fprintf(wp,"sym[%d].name[%d] = '",i,j) ;     */
    /*   fputc(token_buf[j],wp) ;                     */
    /*   fprintf(wp,"' ;\n") ;                        */
         fprintf(wq," 28 %d %d %c",i, j,token_buf[j]);
         fadvance(4) ;
        }
      }
 /*  fprintf(wp,"sym[%d].index = %d;\n",i,symt[symid].descno) ;  */
     fprintf(wq," 29 %d %d",i,symt[symid].descno) ;
     fadvance(3) ;
     i = i + 1 ;
    }
  }
/* printf("%-s\n",buff)  ;  */
/*  fprintf(wo,"sav_write = 0 ;\n") ;        */
 /*  fprintf(wo,"num_print = %d ;\n",i) ;     */
```

```
 /*  fprintf(wp," out_interval = 1 ;\n") ;    */
 fprintf(wp," 30 31 %d 32",i) ;
 fadvance(4) ;
}
/*-------------------------------------------------------------*/


/*-----------------STRING COPYING ROUTINE----------------------*/
strcpy(s,t)             /* copies s = t  */
char *s, *t ;
 {
  while(*s++ = *t++)
   ;
 }
/*-------------------------------------------------------------*/


/*-----------------STRING COMPARISON ROUTINE-------------------*/
/* returns zero if string s is equal to string t               */

strcmp(s,t)
char s[], t[] ;
 {
  int i ;

  i = 0 ;
  while(s[i] == t[i])
    if (s[i++] == '\0')
      return (0) ;
  return(s[i] - t[i]) ;
 }
/*-------------------------------------------------------------*/


/*-------------------GET NEXT ID ROUTINE-----------------------*/
getid(rx, ernm) /* finds the next id and returns it in token_buf */
int ernm ;      /* error number in case of EOF */
FILE *rx ;
 {
  int i , c, flag, j ;

  flag = 0 ;
  delimiter = -1 ;
  i = 0 ;
  while((delimiter < 1 ) || (flag == 0))
   {                      /* flag = 1 when some non blank char is */
                          /* read into token buffer               */
    c = fgetc(rx) ;

    buff[bb] = c ;        /* buff is the 80 character buffer for  */
    bb = bb + 1 ;         /* printing the entire line after it is */
    if (bb > 78)          /* read.     bb = index for buff        */
     {
      if (printflag == 1)
        {
    /*  printf(" %-s\n",buff) ;    */
```

```
        }
      bb = 0 ;
     }
    switch(c)
     {
      case ' ' : deliniter = -1 ;
                 break ;
      case ',' : deliniter = 1 ;
                 break ;
      case ';' : deliniter = 2 ;
                 break ;
      case ':' : deliniter = 3 ;
                 break ;
      case '=' : deliniter = 4 ;
                 break ;
      case ')' : deliniter = 5 ;
                 break ;
      case '(' : deliniter = 6 ;
                 break ;
      case '\n': if (flag == 1)        /* flag = 1 indicates that */
                   deliniter = 7 ;     /* some non blank character*/
                                       /* was read into token_buf */
                 buff[bb-1] = '\0' ;

                 if (prinoflag == 1)
                    {
              /*    printf("%-s\n",buff) ;     print the line and   */
                    }
                                          /* output errors in the */
                 outerror() ;             /* line if any.         */

                 bb = 0 ;                 /* initialize line buff */
                 break ;

      case EOF : printf("%-s\n",buff) ;
                 error(ernm) ;
                 error(33) ;           /* EOF error               */
                 outerror() ;
                 exit(0) ;             /* abort the program       */
                 break ;

      default  : flag = 1 ;
                 deliniter = 0 ;
     }
    if (deliniter == 0 )
     {
      if (i <= 6 )
       {
        token_buf[i] = c ;
        i = i + 1 ;
       }
     }
  }
 token_buf[i] = '\0' ;

}
```

```
/*-------------------------------------------------------------*/


/*---------------FIND_TOKEN PROCEDURE-----------------------------*/
/* Token = maxkey if a nonkeyword name is encountered else it is */
/* equal to the index of the keyword in the keyword table        */

find_token()
 {
  int i ;

  for (i = 0; i < maxkey; i = i + 1)       /* sequential search  */
    if (strcmp (token_buf,keyword[i]) == 0 )
      break ;
  toknn = i ;         /* token = maxkey,   if match is not found  */
 }
/*-------------------------------------------------------------*/


/*---------------SINGLE ID PARSING ROUTINE------------------------*/
parseid(i)
 int i ;                  /* i = token number to be compared to  */
  {
   getid (rp,33) ;               /* find the next identifier */
   find_token() ;                /* find token number        */

   if (toknn == maxkey)          /* check if name != function*/
    {
     findprim() ;
     if (primid < prim_count)
      error(25) ;                /* keyword found            */
    }

   if (toknn != i)               /* identifier of type 'i'   */
    error(i) ;                   /* expected.                */
  }
/*-------------------------------------------------------------*/


/*--------------------FINDID (symbol table index)---------------*/
/* finds the symbol table index. An error message is generated if*/
/* the name does not exist */

findid()
 {
  int i ;

  for (i = 0; i < sym_count; i = i + 1)
    if ( strcmp(token_buf,symt[i].name) == 0)
      break ;

  if (i == sym_count)      /* name not found in the symbol table */
   {
    error(28) ;             /* undeclared name                   */
    symid = -1 ;
   }
```

```
  else
    symid = i ;
 }
/*-------------------------------------------------------------*/


/*---------------FINDDESC (desc table index)-------------------*/
/* This routine is used in conjunction with the DEFINE parsing to*/
/* update all descriptors using the name in sbuf with the decl-  */
/* ared parameters.                                              */
 finddesc(sbuf)
 char sbuf[8] ;
  {
   int i ;
   for (i = lim; i < dptr; i = i + 1)
     if (strcmp(sbuf,desct[i].fun) == 0)
       break ;
    lim = i + 1 ;
    descid = desct[i].dnum ;
   }
/*-------------------------------------------------------------*/


/*----------------------UPDATE DESCTABLE-----------------------*/
/* This routine updates the descriptor table.                    */
/* s = function name (type), num = symbol table index            */
 updesct(s, num)
 char s[8] ;
 int num ;
 {
  strcpy(desct[dptr].fun, s) ;
  desct[dptr].dnum = num ;
  dptr = dptr + 1 ;
 }
/*-------------------------------------------------------------*/


/*----------------FINDPRIM (primitive table index)-------------*/
findprim()
 {
  int i ;
  for (i = 0; i < prim_count; i = i + 1)
    if ( strcmp(token_buf,primt[i].nam2) == 0)
      break ;
  primid = i ;
 }
/*-------------------------------------------------------------*/


/*-----------------SYMBOL TABLE UPDATE-------------------------*/
/* This routine updates the symbol table. sym_count = symbol     */
/* table index, desc_no = descriptor number,                     */
/* typ = function type, 0 = input, -1 = output, -2 = internal    */

update(typ)
```

```
int typ ;
 {
   symt[sym_count].descno = desc_no ;
   symt[sym_count].funcno = typ ;
   strcpy(symt[sym_count].name, token_buf) ;
   sym_count = sym_count + 1 ;
  }
/*-------------------------------------------------------------*/


/*---------------CODE GENERATION (input descriptors)-----------*/
code_input(i)
int i ;                    /* i = desc_no */
 {
 int j;
  if (err_count == 0)
   {
/*  fprintf(wp," desc[%d].pfunc = read_input;\n",i) ;          */
    fprintf(wq," 33 %d 0",i) ;
/*  fprintf(wp," desc[%d].param[0] = %d;\n",i,token_buf[0]) ; */
    j=hashf(token_buf);
    fprintf(wq," 1 %d 0 %d",i, j) ;
              /* parameters 0 and 1 contain the input line name */

/*  fprintf(wp," desc[%d].param[2] = 1;\n",i) ;  */
    fprintf(wq," 1 %d 2 1",i) ;
    fadvance(15) ;
   }
 }
/*-------------------------------------------------------------*/


/*--------------------ERROR MESSAGE ROUTINE--------------------*/
errmessage(i)
int i ;                 /* i = error number */
 {              /* err_ptr = global indicating error table index */
  printf(" ^ERROR^") ;
  switch(i)
   {
    case 0 : printf(" 'MODULE' expected, %s found\n",
                    errt[err_ptr].nm) ;
             break ;
    case 1 : printf(" 'INPUTS' expected, %s found\n",
                    errt[err_ptr].nm) ;
             break ;
    case 2 : printf(" 'OUTPUTS' expected, %s found\n",
                    errt[err_ptr].nm) ;
             break ;
    case 3 : printf(" 'TYPES' expected, %s found\n",
                    errt[err_ptr].nm) ;
             break ;
    case 4 : printf(" '{' expected, %s found\n",
                    errt[err_ptr].nm) ;
             break ;
    case 5 : printf(" '}' expected, %s found\n",
                    errt[err_ptr].nm) ;
```

```
        break ;
case 6 : printf(" 'INITIALIZE' expected,%s found\n",
                 errt[err_ptr].nm) ;
        break ;
case 7 : printf(" 'PRINTOUT' expected, %s found\n",
                 errt[err_ptr].nm) ;
        break ;
case 9 : printf(" 'DEFINE' expected, found ,%s \n"
                 ,errt[err_ptr].nm) ;
        break ;
case 25: printf(" name %s is a keyword\n",
                 errt[err_ptr].nm) ;
        break ;
case 26: printf(" count = %d, >>COMPILATION discontinued\n",
                                                 err_count);
        break ;
case 27: printf(" '=' expected after %s\n",
                 errt[err_ptr].nm) ;
        break ;
case 28: printf(" %s is undeclared\n",
                 errt[err_ptr].nm) ;
        break ;
case 29: printf(" '(' expected after %s\n",
                 errt[err_ptr].nm) ;
        break ;
case 30: printf(" %s is undefined function\n",
                 errt[err_ptr].nm) ;
        break ;
case 31: printf(" ',' expected after %s\n",
                 errt[err_ptr].nm) ;
         break ;
case 32: printf(" ')' expected after %s\n",
                 errt[err_ptr].nm) ;
         break ;
case 33: printf(" unexpected EOF\n");
         break ;
case 34: printf(" missing END\n") ;
        break ;
case 35: printf(" incorrect # of args. on LHS %s\n"
                , errt[err_ptr].nm) ;
        break ;
case 36: printf(" in syntax, %s unrecognized \n",
                 errt[err_ptr].nm) ;
        break;
case 37: printf(" count = 10, >>Compilation discontinued\n");
        break ;
case 38: printf(" = 0,  ***END OF COMPILATION***\n") ;
        break ;
case 39: printf(" 1st DELAY index is > lim\n");
        break ;
case 40: printf(" 2nd DELAY index is > lim\n");
        break ;
case 41: printf(" missing (\n") ;
        break ;
case 42: printf(" missing INITIALIZE\n") ;
        break ;
```

```
    case 43: printf(" missing ';'\n") ;
             break ;
    case 44: printf(" undefined function\n") ;
             break ;
    case 45: printf(" missing TYPES\n") ;
             break;
    case 46: printf(" missing }\n") ;
             break ;
    case 47: printf(" missing ')'\n");
             break;
    case 48: printf(" missing DEFINE\n");
             break ;
    case 49: printf(" incorrect # of inout arguments in call\n");
             break ;
    case 50: printf(" incorrect # of out arguments in call\n");
             break ;
   }
  err_count = err_count + 1 ;
  if (err_count > 9)
   {
    error(37) ;
    outerror() ;
    exit(0) ;
   }
 }
/*-------------------------------------------------------------*/


/*-------------------------OUTERROR ROUTINE--------------------*/
/* This routine outputs all errors encountered in a line after   */
/* entire line has been read.                                    */

 outerror()
  {
   int i;
   i = 0 ;
   while (err_ptr >= 0)      /* if error count for a line is > 0 */
    {                        /* print all errors encountered     */
     errmessage(errt[i].errno) ;
     i = i + 1 ;
     err_ptr = err_ptr - 1 ;
    }
  }
/*-------------------------------------------------------------*/


/*---------------------ERROR ROUTINE---------------------------*/
/* This routine enters the error number and the name of the wrong*/
/* identifier in the errt (error table). The errors are printed  */
/* after the whole line has been scanned.                        */

 error(i)
  int i ;       /* i = error number */
  {
   err_ptr = err_ptr + 1 ;          /* error count for one line  */
   errt[err_ptr].errno = i ;        /* errno = error number      */
```

```
   strcpy(errt[err_ptr].nm,token_buf) ;
                                   /* copy name of wrong identi.*/
  }
/*-------------------------------------------------------------*/


/*-----------------firstp()------------------------------------*/
/* This routine scans the user program for any expansion requests*/
/* if found, they are put on a expt stack along with their func #*/
/* Note that func no. may not be known at this time             */
 firstp()
 {
 extern int maxpid;
 int i,j,option,tpid,found;
 char temp1[8], target[8];


 ri=fopen("pil","r");
  search(ri, ri, -1, 9, 0,48) ; /* search for DEFINE (9)     */
                                /* 48 = missing DEFINE error */
  getid(ri,42) ;       /* 42 = missing INITIALIZE error */
  find_token() ;
  while (tokno!=6)        /* search for EXPAND */
     {
     if (tokno == 21)     /* EXPAND */
        {
        sfound=0;
        getid(ri,33) ;         /* this is what we expand;  33 = EOF error */
        strcpy(expt[expcount].fname, token_buf) ;
        strcpy(temp1,token_buf);       /* save name for later */
        expcount++;
        getid(ri,33);        /* how far do we expand? */
        findprim();
        if (primid<prim_count)    /* a valid primitive? */
           {
           strcpy(target,primt[primid].nam2);    /* yes, save it as is */
           tpid=primid;
           }
        else
           {
           found=0;                                /* no, what type is it? */
           for (i=0; i<=maxpid; i++)
              {
              j=0;
              while (typelist[i][j].used==1)
                 {
                 if (strcmp(token_buf,typelist[i][j].sname)==0)  /* match? */
                    {
                    strcpy(target,primt[i].nam2);   /* primitive name */
                    tpid=i;                         /* primitive id */
                    found=1;
                    break;
                    }
                 else
                    j++;
                 }  /* end while */
```

```
            if (found==1)
               break;
            }     /* end for */

         if ((i>maxoid) && (found==0))
            error(44);
         }        /* end else */

      check_deeper(temp1,tpid,target);
      if (sfound==0)
         {
         printf("I couldn't find %s in any of the circuit descriptions.\n",
                                             target);
         printf("\n\nDo you wish to: \n\n");
         printf("   1.  Continue anyway\n");
         printf("   2.  Give up\n\n");
         printf("Enter the number of your selection ");
         scanf("%d",&option);
         if (option==2)
            exit(0);
         }

      }  /* end if token=21 */
   getid(r1,42);                  /* look for INITIALIZE */
   find_token();
   }                              /* end while token */

multi_mod();                        /* do multimodule case after  */
fclose(r1);                         /* enumerated EXPANDs         */
}
/*-------------------------------------------------------------*/


/*-------------------------secondp()-----------------------------*/
/* second pass routine, expansion is carried out in this routine */
/* First the specs for user program are copied to P1EXP. The func*/
/* n for function to be expanded is determined at the same time. */
/* Next, the function's description is copied to SCR1. The user  */
/* program is searched for any call to funtion to be expanded,   */
/* The code from SCR1 is substituted at this point. The expanded */
/* circuit is stored in SCR2. Finally P1EXP, SCR2 are appended   */
/* along with the simulation specs from user program.            */

secondp(expnum)
int expnum ;        /* expnum = expand table index */
{
 int i, j ;
 count = 0 ;
/*--search for TYPES in user prog (copy to P1EXP)---------------*/
 search( r1, w1, -1, 3, 1,45) ; /* 45 = missing TYPES error     */

/*----------search if function to be expanded is a TYPE----------*/
 while (1)          /* search function to be expanded. Replace it*/
  {                 /* with x's. Find its fnum and put in expt   */
   getid(r1,41) ;   /* 41 = missing ( error */
   find_token() ;
```

```
    if (toknn == 4 )        /* '(' then break */
      break ;
    pdelim(w1) ;            /* print to P1EXP */

    if (toknn != 8)         /* function name, not an INTERNAL */
     {
      findprim() ;
      if (primid >= prim_count)
       error(30) ;

      while (1)             /* search function's name in TYPE list */
       {
        getid(r1,43) ;      /* 43 = missing ; error */
        if (strcmp(token_buf, expt[expnum].fname) == 0)
         {
          expt[expnum].fnum = primid ;
          strcpy(token_buf,"XXXXXXX") ;
         }
        pdelim(w1) ;
        if (delimiter == 2)     /* if ';' then end of TYPE list   */
          break ;
       }                    /* end type search */
     }
    else     /* INTERNALS */
      {
      search(r1, w1, 2, -1, 1,43);/* print internals as is */
                                  /* 2 = delim ';'  (error 43)*/
      }
   }       /* end TYPES */
/*-----------------find primid for the function-----------------*/
  if (expt[expnum].fnum == -1)/* if no types,then find function **/
   {
    strcpy(token_buf,expt[expnum].fname) ;
    findprim() ;
    if (primid >= prim_count)
      error(30) ;
    else
      expt[expnum].fnum = primid ;
   }
/*-------------------------------------------------------------*/
  fclose(r1) ;
/*---------------------find function in lib---------------------*/
  r2 = fopen("lib1","r") ;     /* library */
  s1 = fopen("scr1","w") ;      /* scratch file for function */

  while (1)        /* find the function to be expanded in the lib */
   {
    getid(r2,44); /* 44 = function not found in lib */
    find_token() ;
    if (toknn == 0)     /* MODULE */
     {
      getid(r2,33) ;
      if (strcmp(token_buf, print[(expt[expnum].fnum)].nam2) == 0)
        {
        break ;
        }
```

```
    }
  }
/*-----------------function found-----------------------------*/
/*--------store inputs, outputs on  stacks---------------------*/
  getid(r2,33) ;          /* INPUTS */
  incount = 0 ;
  while (1)
   {
    getid(r2,43) ;        /* 43 = missing ; error */
    find_token() ;
    if (tokno == 2)       /* OUTPUTS */
      break ;
    else
     {
      strcpy(instack[incount],token_buf) ;
      incount = incount + 1 ;
     }
   }
  outcount = 0 ;
  while (1)    /* OUTPUTS */
   {
    getid(r2,45) ;   /* 45 = missing TYPES error */
    find_token() ;
    if (tokno == 3)  /* TYPES */
      break ;
    else
     {
      strcpy(outstack[outcount],token_buf) ;
      outcount = outcount + 1 ;
     }
   }
/*---------------TYPES should be tacked by function name----------*/
/*---------------save types on a typstack-------------------------*/
  typcount = 0 ;
  while (1)
   {
    getid(r2,41) ;       /* 41 = missing { error */
    find_token() ;
    if (tokno == 4)      /* {         */
      break ;
    else
     {
      if (tokno == 8)    /* INTERNALS */
       {
        pdelim(s1) ;           /* internals are copied on to SCR1*/
        search(r2,s1,2,-1,1,43);/* so that they can be reproduced*/
                               /* 'occurance' times at the end   */
       }
      else /* TYPE declarations */
       {
        pdelim(w1) ;   /* function types are tacked by the name  */
        while (1)      /* of the function to be expanded         */
         {             /* They are saved on a stack so that it   */
                       /* can be determined in its desc, if a    */
                       /* type is used (will be tacked similarly)*/
          getid(r2,43) ;  /* 43 = missing ; error */
```

```
            strcpy(typstack[typcount], token_buf) ;
            typcount = typcount + 1 ;

            for (i = 0; i < 5; i = i + 1)
             {
              if (token_buf[i] == '\0')
                break ;
             }
            for (j = i; j < 5; j = j + 1)
              token_buf[j] = ' ' ;
            token_buf[5] = printf[expt[expnum].fnum].nam2[0] ;
            token_buf[6] = printf[expt[expnum].fnum].nam2[1] ;
            token_buf[7] = '\0' ;
            pdelim(w1) ;
            if (delimiter == 2)      /* ; */
              break ;
           }
         }
       }
     }

  fprintf(s1," { \n") ;
  count = 0 ;
/*-------------------------------------------------------------*/

/*-------------write structural desc. to SCR1-------------------*/
  while (1)
   {
    search(r2, s1, 4, 5, 1,46);/* 4 = '=' , 5 = '}'       */
              /* write structural description to scr file */
              /* Types are tacked by function's name */
    if (token== 5)
      break ;

    getid(r2,46) ;/* 46 = missing } error    */
    ftype() ;
    if (ft != 0)  /* indicates type declared */
     {
      for (i = 0; i < 5; i = i + 1)
       {
        if (token_buf[i] == '\0')
         break ;
       }
      for (j = i; j < 5; j = j + 1)
        token_buf[j] = ' ' ;
      token_buf[5] =  printf[expt[expnum].fnum].nam2[0] ;
      token_buf[6] =  printf[expt[expnum].fnum].nam2[1] ;

      token_buf[7] = '\0' ;
     }
    pdelim(s1) ;
    search(r2, s1, 5, -1, 1,47);  /* 5 = ')' */
                                  /* outputs */
    count = 0 ;
   }
  fclose(w1) ;
```

```
  fclose(s1) ;
  fclose(r2) ;
  count = 0 ;
  cexpand(expnum);/*band the primitive whenever it occurs in ckt */
 }
/*-------------------------------------------------------------*/


/*---------------------pdelim()-------------------------------*/
/* prints delimiter on the file */
 pdelim(wx)
 FILE *wx ;
 {
  fprintf(wx," %s",token_buf) ;
  count = count + 1 ;
  switch(delimiter)
   {
    case 8 : fprintf(wx,"  ") ;
             break ;
    case 7 : fprintf(wx," \n") ;
             count = 0 ;
             break ;
    case 6 : fprintf(wx," (") ;
             break ;
    case 5 : if (print_select==0)
                {
                fprintf(wx," )");
                break;
                }
             else
                {
                fprintf(wx," ) ; \n");
                count=0;
                break;
                }
    case 4 : fprintf(wx," =") ;
             break ;
    case 3 : fprintf(wx," :") ;
             break ;
    case 2 : fprintf(wx," ;\n") ;
             count = 0 ;
             break ;
    case 1 : fprintf(wx," ,") ;
             break ;
   }
  if (count >= 7)
   {
    fprintf(wx," \n") ;
    count = 0 ;
   }
 }
/*-------------------------------------------------------------*/


/*------------------fcopy()--------------------------------*/
fcopy(rr, ww)
```

```
FILE *rr, *ww ;
{
 int c ;
 while ((c = getc(rr)) != EOF)
   putc(c,ww) ;
}
/*-------------------------------------------------------------*/


/*------------------copy_noend()-------------------------------*/
copy_noend(inf,outf)
FILE *inf,*outf;
{
int pdone;

   pdone=0;
   print_select=1;
   getid(inf);                          /* read the token from file */
   find_token();                        /* see what it is           */
   while (pdone==0)                     /* quit when we see END     */
      {
      switch (toknn)
         {
         case 3:   pdelim(outf);        /* write the TYPES token */
                   getid(inf,33);       /* and get the next one  */
                   find_token();
                   if (toknn==4)        /* check for no TYPES    */
                      {
                      fprintf(outf," ; \n");  /* no types, start new line */
                      }
                   break;

         case 6:   print_select=1;
                   pdelim(outf);         /* write the INITIALIZE token */
                   getid(inf,33);        /* check for sequence of */
                   find_token();         /* INITIAL;; PRINTOUT:   */
                   if (toknn==7)         /* is this PRINTOUT?     */
                      fprintf(outf," ; \n");  /* yes, start new line */
                   break;

         case 9:   print_select=0;
                   pdelim(outf);         /* write the DEFINE token */
                   getid(inf,33);        /* check for sequence of */
                   find_token();         /* DEFINE:; INITIAL:; PRINTOUT: */
                   if (toknn==6)         /* is this INITIAL? */
                      {
                      fprintf(outf," ; \n");  /* yes, print new line */
                      pdelim(outf);           /* then print INITIAL  */
                      getid(inf,33);
                      find_token();
                      if (toknn==7)           /* is this PRINTOUT? */
                         fprintf(outf," ; \n"); /* yes, print new line */
                      }
                   break;

         case 20:  pdone=1;
```

```
                    break;

          default:  pdelim(outf);          /* write the token */
                    getid(inf,33);         /* and get the next one */
                    find_token();
                    break;
          }    /* end switch (toknn) */
       }    /* end while (toknn) */
   print_select=0;
}
/*-------------------------------------------------------------*/

/*----------------------REVERSE()------------------------------*/
reverse (s)
char s[] ;
 {
  int c, i, j ;
  for (i = 0, j = 6; i < j; i++, j--)
   {
    c = s[i] ;
    s[i] = s[j] ;
    s[j] = c ;
   }
 }
/*-------------------------------------------------------------*/


/*---------------------------itoa()----------------------------*/
itoa(n, s)
char s[] ;
int n ;
{
 int i ;

 i = 0 ;
 do {
     s[i++] = n % 10 + '0' ;
    } while ((n /= 10) > 0) ;
   reverse(s) ;
}
/*-------------------------------------------------------------*/


/*--------------------FTYPE()----------------------------------*/
/* It finds if a type for the function name in the primitive's   */
/* description has been declared. ft = 1 indicates this         */
 ftype()
  {
   int i ;
   ft = 0 ;
   for (i = 0; i < typcount; i = i + 1)
    {
     if (strcmp(token_buf, typstack[i]) == 0)
      {
       ft = 1 ;
       break ;
```

```
      }
    }
  }
/*-------------------------------------------------------------*/


/*---------------------cexpand()-------------------------------*/
 cexpand(exnum) /* expand primitive's occurance */
 int exnum ;    /* exnum = expand table index   */
 {
  int i ;

  s2 = fopen("scr2","w") ;     /* expanded desc */
  r1 = fopen("p11", "r") ;     /* user prog     */
  occurance = 0 ; /* # of times call to function is made */
                  /* internals will be duplicated this # */

  search(r1,r1,-1,4,0,41);    /* find circuit description */
                              /* 4 =  { */
/*-------------------------------------------------------------*/
  skip = 0 ;
  while (1)
   {
    ckt_line(&outicount, r1,outistack, &inicount, inistack, &skip) ;

    if (found_end==1)              /* find an ENDSWAP while in cktline?*/
       {
       found_end=0;                /* sure did, set up for the next one*/
       fprintf(s2," ENDSWAP ; \n");  /* write the keyword onto SCR2  */
       }
    if (found_start==1)            /* find a SWAPLIN while in cktline? */
       {
       found_start=0;              /* yes, get ready for the next one */
       fprintf(s2," SWAPLIN ;\n"); /* write the keyword onto SCR2     */
       }

    if (skip == 1)
      break ;
/*-------------------------------------------------------------*/

/*--------------------print same, or substitute code-----------*/
    if (strcmp(savfunc, expt[exnum].fname) == 0)
     {
      if (outicount != outcount)
        error(50) ;
      if (inicount != incount)
        error(49) ;

      if (err_count == 0)
       {
        substitute() ;    /* substitute the primitive's code */
        occurance = occurance + 1 ;
       }
     }
    else                /* print the code as is            */
     {
```

```
      for (i = 0; i < outicount ; i = i + 1)
       {
        strcpy(token_buf, outistack[i]) ;
        delimiter = 1 ;   /* , */
        if (i == (outicount-1))
          delimiter = 4 ; /* =    */
        odelim(s2) ;      /* SCR2 */
       }

      strcpy(token_buf, savfunc) ;
      delimiter = 6 ;    /* ( */
      odelim(s2) ;

      for (i = 0; i < inicount; i = i + 1)
       {
        strcpy(token_buf, inistack[i]) ;
        delimiter = 1 ;
        if (i == (inicount-1))
          delimiter = 5 ; /* ) */
        odelim(s2) ;
       }
      count = 0 ;
      fprintf(s2," ;\n") ;
     }   /* end else */
   }
  fclose(s2) ;
 }
/*-------------------------------------------------------------*/


/*----------------------substitute()----------------------------*/
/* copies function's code from s1 to s2. Internals are tacked by */
/* occurance number */
 substitute()
 {
  int i, skip1 ;

  s1 = fopen("scr1", "r") ;   /* function's description */
  search(s1,s1,-1, 4, 0,41) ;      /* find description */
                                   /* 4 = '('          */
 /*-------------------------OUTPUTS-------------------------------*/
  skip1 = 0 ;
  while (1)
   {
    ckt_line(&out2count, s1, out2stack, &in2count, in2stack,&skip1) ;

    if (skip1 == 1)
      break ;

/*--------------------write to SCR2 (s2)-------------------------*/
/*inputs/outputs are replaced by corresponding names from in/out1*/
    for (i = 0; i < out2count; i = i + 1)
     {
      foutorder(out2stack[i]) ;
      if (outorder != -1)
        {
```

```
        strcpy(token_buf, out1stack[outorder]) ;
        }
      else
       {
        finorder(out2stack[i]) ;
        if (inorder != -1 )
          {
          strcpy(token_buf, in1stack[inorder]) ;
          }
        else
          tack(occurance,out2stack[i]) ;
       }
      delimiter = 1 ;
      if (i == (out2count - 1))
        delimiter = 4 ;        /* = */
      pdelim(s2) ;
     }

    strcpy(token_buf, savfunc) ;
    delimiter = 6 ;    /* ( */
    pdelim(s2) ;

    for (i = 0; i < in2count; i = i + 1)
     {
      finorder(in2stack[i]) ;
      if (inorder != -1)
        {
        strcpy(token_buf, in1stack[inorder]) ;
        }
      else
       {
        foutorder(in2stack[i]) ;
        if (outorder != -1)
          {
          strcpy(token_buf, out1stack[outorder]) ;
          }
        else
          tack(occurance,in2stack[i]) ;
       }
      delimiter = 1 ;
      if (i == (in2count - 1))
        delimiter = 5 ;   /* )  */
      pdelim(s2) ;
     }
    fprintf(s2," ;\n") ;
    count = 0 ;
   }   /* end while */
  fclose(s1) ;
 }
/*-------------------------------------------------------------*/


/*-----------------foutorder()---------------------------------*/
 foutorder(s)
 char s[] ;
  {
```

```
   int i ;
   for (i = 0; i < outcount; i = i + 1)
    {
     if (strcmp(s, outstack[i]) == 0)
       break ;
    }
   if (i >= outcount)
    outorder = -1 ;
   else
    outorder = i ;
  }
/*-------------------------------------------------------------*/


/*-----------------finorder()----------------------------------*/
 finorder(s)
 char s[] ;
  {
   int i ;
   for (i = 0; i < incount; i = i + 1)
    {
     if (strcmp(s, instack[i]) == 0)
       break ;
    }
   if (i >= incount)
    inorder = -1 ;
   else
    inorder = i ;
  }
/*-------------------------------------------------------------*/


/*----------------------ckt_line()-----------------------------*/
/* This routine reads one line of circuit description and stores */
/* inputs, outputs in proper arrays                              */
ckt_line(oocount, rx, oostack, iicount, iistack, skip1)
int *oocount, *iicount, *skip1 ;
FILE *rx ;
char oostack[][8], iistack[][8] ;
 {
                    /* Outputs */
  *oocount = 0 ;
  while (1)       /* put outputs on out stack */
   {
    getid(rx,46);/* 46 = missing } error     */
    find_token() ;

    if (toknn==24)                        /* did we find an ENDSWAP?*/
       {
       found_end=1;                       /* yes, tell CEXPAND      */
       getid(rx,46);                      /* get the next token     */
       find_token();                      /* and see what it is     */
       }
    if (toknn==23)                        /* find a SWAPLIN too?    */
       {
       found_start=1;                     /* yes, tell CEXPAND      */
```

```
        getid(rx,46);
        find_token();
        }

    if (toknn == 5)  /* ) */
     {
      *skip1 = 1 ;
      break ;
     }
    strcpy(oostack[*oocount], token_buf) ;
    *oocount = *oocount + 1 ;
    if (delimiter == 4)   /* = */
      break ;
   }     /* end while outputs */
/*-------------------------------------------------------------*/
  if ((*skip1) != 1)
   {
    getid(rx,33) ;            /* function name */
    strcpy(savfunc, token_buf) ;

/*---------------------------INPUTS----------------------------*/
    *iicount = 0 ;
    while(1)  /* inputs */
     {
      getid(rx,47) ;  /* 47 = missing ')' error */
      strcpy(iistack[*iicount], token_buf) ;
      *iicount = *iicount + 1 ;
      if (delimiter == 5)    /* ) */
        break ;
     }
   }
 }
/*-------------------------------------------------------------*/


/*-------------------search()------------------------------------*/
search(xi, xo, delm, tokm, fq,ernum)
FILE *xi, *xo ;
int delm, tokm, fq, ernum ;
{
 while (1)
  {
   getid(xi,ernum) ;
   if (fq == 1)
    {
    pdelim(xo) ;
    if (delimiter == 5)
     {
      fprintf(xo," ;\n") ;
      count = 0 ;
     }
    }
   if (delimiter == delm)
     break ;
   find_token() ;
   if (toknn == tokm)
```

```
      break ;
   }
}
/*-------------------------------------------------------------*/


/*-----------------------tack()--------------------------------*/
tack(occ,iobuff)
int occ ;
char iobuff[8] ;
{
 int i, j ;
 strcpy(token_buf, iobuff) ;

 for (i = 0; i < 7; i = i + 1)
  {
   if (token_buf[i] == '\0')
     break ;
  }
 for (j = i; j < 7; j = j + 1)
  token_buf[j] = ' ' ;
 token_buf[7] = '\0' ;
 reverse(token_buf) ;
 itoa(occ, token_buf) ;
}
/*-------------------------------------------------------------*/


/*--------------------multi_mod()------------------------------*/
/* This routine handles sub modules. All sub-modules are tacked  */
/* to the library (STRUC) in the front. Also expand table is     */
/* incremented for each sub-module.                              */
multi_mod()
{
 lii = fopen("libi","w") ;
 while(1)
  {
   getid(ri,34) ;     /* 34 = missing END error */
   find_token() ;
   if (toknn == 20)  /* END */
     break ;
   if (toknn == 0)   /* MODULE */
    {
     pdelim(lii) ;
     getid(ri,33) ;  /* sub module name */
     strcpy(primt[prim_count].nam2, token_buf) ;
     strcpy(expt[expcount].fname, token_buf) ;
     expt[expcount].fnum = prim_count ;
     prim_count = prim_count + 1 ;
     expcount = expcount + 1 ;
     pdelim(lii) ;
     search(ri, lii, -1, 5, 1,46); /* 5 = } */
    }
  }
ti = fopen("struc","r") ;
 fcopy(ti,lii) ;
```

```
 fclose(t1) ;
 fclose(r1) ;
 fclose(lii) ;
}
/*-----------------------------------------------------------------*/


/*---------------------fadvance()-----------------------------*/
 fadvance(numm)
  int numm ;
  {
   filecount = filecount+  numm ;
   if (filecount > 22)
    {
     filecount = 0 ;
     fprintf(wq," \n") ;
    }
  }
/*-----------------------------------------------------------------*/


/*-------------------------hashf()-------------------------------*/
hashf(s)    /* finds hash value for string s  */
char *s;
{
 int hashval ;
 for (hashval = 0; *s != '\0' ;)
    hashval += *s++ ;                /* convert from string to # */

 test(&hashval);                     /* and test for collision */

 hashtable[hashcount]=hashval;
 hashcount++;

 return (hashval) ;
}
/*-----------------------------------------------------------------*/


/*-------------------------------test()----------------------------*/
test(value)
int *value;
{
   int i;
   for (i=0; i<hashcount; i++)
      {
      if (hashtable[i]==(*value))          /* hashtable collision? */
         {
         (*value)=(*value)+11;             /* yes, add a prime number..*/
         test(value);                      /* and test again         */
         }
      }
}
/*-----------------------------------------------------------------*/
```

```
/*****************************************************************
 *                                                               *
 *                  Precompilation Routines                      *
 *                     for the compiler                          *
 *          including:  structural expansion handling            *
 *                      USING handling                           *
 *****************************************************************/

#include "stdio.h"              /*<stdio.h> in UNIX environment   */


#define maxkey 29               /* maximum number of keywords      */

#define maxprim 100             /* maximum number of primitives    */

#define maxouts 32              /* maximum of 32 outputs per prim  */

/*------------------GLOBAL DECLARATIONS-----------------------------*/
extern int strcpy() ;       /*   copies one string to another */

extern int strcmo() ;       /*   string comparison                */

extern int getid()  ;       /*   get next identifier          */

extern int find_token() ;   /*   returns the token number     */
                            /*   (keyword table index)        */
                            /*   the given function name/type */

extern int findprim() ;     /*   finds primitive library index*/

extern int pdelim() ;       /*   prints a file of keywords and */
                            /*   tokens, separated by delimiters*/

extern int fcopy() ;        /*   file copying routine         */

extern int copy_noend();    /*   copies everything but END token */

extern int ckt_line() ;     /*   scans one line of ckt desc.  */

extern int search() ;       /*   search a delimiter or token  */

extern int multi_mod();

extern int secondp();

extern int tack();

/*---------------------------------------------------------------*/


/*----------------DATA STRUCTURES-------------------------*/

struct prim_tab {           /*  Primitive table :           */
  char nam[8] ;             /*  primitive table             */
  char nam2[8] ;            /*  EXTENDED primitive table    */
```

```
  int numpar, outp ;        /*  numpar = no. of parameters  */
  int normld, fanout ;      /*  for the function            */
  int technology, overld ;  /*  outp = # of outputs         */
 } ;

struct namepair {           /* this holds system-generated  */
  char e_from[8];           /* expansion requests           */
  char e_to[8];
 };

struct swaoname {           /* each of these nodes will contain a module */
  char sname[8];            /* specified in the USING parameter list */
  int used;                 /* this indicates whether used or not    */
  };

struct functable {          /* each of these nodes will contain a function*/
  char fnname[8];           /* and level for a library VOHL module   */
  int level;
  };

struct exp_tab {            /* expansion table  */
  char fname[9];
  int fnum;
  };
/*-----------------------------------------------------------*/


/*---------------STORAGE ALLOCATION-------------------------*/
extern struct prim_tab primt[maxprim], *primptr ;
extern struct swaoname typelist[maxprim][8];  /* table of replacements*/
extern struct swapname swaplist[20][8];  /* USING parameter list storage */
extern struct namepair reqtable[maxprim];
extern struct exp_tab exot[30];

extern char userprg[8];
extern int expcount;
extern int sfound;
extern int req_count;       /* number of system expand reqs */
extern int line_count;      /* line index for swaplist      */
int line_item;              /* index within single line     */
extern int swap_flag;       /* indicates whether all of this*/
                            /* is necessary or not          */
int expand_flag;            /* indicates need for additional*/
                            /* expansion declarations       */
int found_start;            /* indicates presence of SWAPLIN*/
int found_end;              /* indicates presence of ENDSWAP*/

extern int add_flag;        /* set if cell is to be added to*/
                            /* primitive library            */
extern int delimiter, bb, skip ;  /* delimiter = delimiter type   */
                            /* bb = buff[80] index (line)   */
extern int tokno ;      /* err_count  = error count     */
extern int cellcount;               /* # of MODULEs in user program */
extern int prinpflag;       /* controls printing of source  */
extern int print_select;    /* determines whether a ')' causes*/
                            /* a linefeed or not (default=no)*/
```

```
extern int prim_count, primid ;   /* prim_count = # of primitives */
extern int sys_prims;             /* number of system-defined prims*/

extern int features[maxprim][2];     /* used to describe the type of */
                           /* descriptions available;           */
                           /* -1 -> empty     0 -> block only   */
                           /* 1 -> struc only  2 ->block & struc */

extern int append_table[maxprim]; /* keeps track of the functions we've */
extern int append_index;   /* added to the user program          */
int modules_to_do;         /* number of STRUC-only additions to do */
extern int expdone ;       /* marks completion of STRUC expansion*/
extern int no_compilation; /* used when only making library additions*/
extern char token_buf[8], savbuf[8], buff[80] ; /* buff = 1 line  */
extern char keyword[maxkey][8] ;  /* keyword table               */
extern char instack[maxouts][8], outstack[maxouts][8] ;

extern int incount , outcount ;
extern int count ;           /* for printing on the file */
extern char savfunc[8] ;
int maxpid;

extern FILE *r1 ;            /* pointer to input data file        */
extern FILE *r2 ;            /* pointer to STRUCT library         */
extern FILE *r3 ;
extern FILE *r4 ;
extern FILE *w1 ;            /* pointer to expanded file   P1EXP  */
extern FILE *t1 ;
extern FILE *s1 ;
extern FILE *s2 ;

/*-------------------------------------------------------------*/


/*---------------------- countcells() -------------------------*/
countcells(lookfile)
FILE *lookfile;
{
   int start_looking;
   start_looking=0;

   getid(lookfile,33);
   find_token();
   if ((toknn>25) && (toknn<maxkey))     /* first keyword an add key?*/
      start_looking=1;                   /* yes, start looking for   */
                                         /* a missing DEFINE.        */
   while (toknn!=5)                      /* take a look at the 1st   */
      {                                  /* module...                */
      if (toknn==0)                      /* find a MODULE token? */
         cellcount++;                    /* yes, update count    */
      getid(lookfile,33);
      find_token();
      }

   getid(lookfile,33);                   /* read the next token      */
```

```
   find_token();                          /* should be a DEFINE       */
   if ((toknn!=9) && (start_looking==1)) /* is it?  if not...        */
      {
      no_compilation=1;                   /* this is a cell addition only.*/
      printf("no compilation this pass\n"); /* note--if this is just a  */
      }                                   /* missing DEFINE, let Ausif*/
                                          /* handle it. After all, he */
                                          /* wrote this thing <grin>. */
   while (toknn!=20)                      /* scan the rest of the ckt */
      {
      if (toknn==0)                       /* find a MODULE token? */
         cellcount++;                     /* yes, update count    */
      getid(lookfile,33);
      find_token();
      }
}
/*-------------------------------------------------------*/

/*---------------------- build() --------------------------------*/
build()                              /* build a VOHL file with no */
   {                                 /* END keyword               */
   FILE *adds,*specs;                /* modules added go here     */
   int adflag, current_primitive;    /* used locally for each module */
   int i,j,skip1,done;
   int found, divert;
   char buffer1[8];                  /* temporary string buffer   */

   add_flag=0;
   adflag=0;
   done=0;
   divert=0;
   maxpid=0;                         /* largest primid encountered */
   print_select=1;                   /* newline after ')'         */
   adds=fopen("newckts.vhl","w");
   specs=fopen("specs","w");
   getid(r1,33);
   find_token();
   while (done==0)
      {
      if ((toknn>25) && (toknn<maxkey))
         {
         add_flag=1;
         adflag=1;
         pdelim(adds);               /* save the keyword in addition file*/
         getid(r1,33);
         find_token();
         }


      switch(toknn)
         {
         case 0:   divert=0;          /* save MODULE keyword     */
                   pdelim(r2);
                   if (adflag==1)
                      {
                      count--;
```

```
            odelim(adds);   /* and to the new ckt file      */
            }
         getid(r1,33);
         find_token();
         break;

case 3:  pdelim(r2);         /* save the TYPES keyword       */
         if (adflag==1)
            {
            count--;
            pdelim(adds);   /* and to the new ckt file      */
            }

         while (1)
            {
            getid(r1,33);
            find_token();
            if ((toknn==8) || (toknn==4))      /* quit for INTERNA or
               break;

            pdelim(r2);
            find_token();       /* find out what we just read  */
            findprim();         /* it might be a prim, so check */

            if (primid < prim_count)     /* if this is a primitive */
               {
               current_primitive=orimid; /* next IDs will be of this t
               if (primid>maxpid)
                  maxpid=orimid;
               }

            else
               {                         /* not a primitive, so add it
               i=0;
               while(typelist[current_primitive][i].used==1)
                  i++;                   /* look for next free space*/
               strcpy(typelist[current_primitive][i].sname,token_buf);
               typelist[current_primitive][i].used=1;
               }
            }  /* while (1) */

         if (toknn==4)
            {
            count=0;             /* if a '(', then no TYPES */
            fprintf(r2," ; \n"); /* so we need to save a ';' */
            if (adflag==1);
               fprintf(adds, " ; \n");
            pdelim(r2);           /* write the '(' token   */
            if (adflag==1)
               pdelim(adds);
            getid(r1,33);
            find_token();
            }

         break;
```

```
case 5: if (adflag==1)    /* if token is a ')' then */
           {                        /* that's the end of this */
           adflag=0;               /* module, so print ')' and */
           fprintf(adds," ) \n");   /* clear the addition flag*/
           }
        pdelim(r2);
        getid(r1,33);
        find_token();
        break;

case 6:  uexpand();            /* print system expand reqs  */
         print_select=1;       /* put linefeeds after ')'s  */
         divert=0;
         pdelim(r2);           /* print INITIAL             */
         getid(r1,33);
         find_token();
         if (toknn==7)
            fprintf(r2," ; \n ");
         break;

case 9:  print_select=0;      /* no linefeed after ')'     */
         pdelim(r2);          /* print DEFINE token        */
         getid(r1,33);        /* read the next token       */
         find_token();
         switch(toknn)        /* anything to DEFINE?      */
            {
            case 6: print_select=1;  /* no, INITIAL found  */
                    divert=0;
                    fprintf(r2," ; \n");  /* start a new line */
                    uexpand();
                    pdelim(r2);      /* and print INITIAL */
                    getid(r1,33);
                    find_token();
                    if (toknn==7)    /*  no INITIALized params */
                       fprintf(r2," ; \n");
                    break;

            case 21: divert=0;       /* user wants EXPANDs   */
                     fprintf(r2," ; \n");  /* start a new line     */
                     uexpand();      /* put ours first though*/
                     pdelim(r2);     /* now print user's     */
                     getid(r1,33);
                     find_token();
                     break;

            default: divert=1;       /* DEFINE params present */
                     fprintf(r2," ; \n");  /* start new line */
                     fprintf(specs,"DEFINE: ");
                     pdelim(specs);
                     getid(r1,33);
                     find_token();
                     break;
            }
         break;
```

```
case 20: done=1;              /* END token, so quit       */
         break;

case 21: divert=0;            /* EXPAND   */
         pdelim(r2);
         getid(r1,33);
         find_token();
         break;

case 22: line_item=0;         /* USING-- here we go.      */
                              /* temporary file           */
         fprintf(r2,"SWAPLIN ; \n");  /* mark this ckt line */
         swap_flag=1;                 /* we'll be doing swaps */
         expand_flag=1;               /* and probably calling */
         found=0;
         do                           /* for expansions       */
            {
            getid(r1,47);             /* get the parameter list */
            find_token();
            if (toknn==25)            /* is this a NOEXP?   */
               {
               expand_flag=0;         /* don't need to call for*/
               getid(r1,47);          /* expansions            */
               }
            else                      /* expand to this TYPE */
             {
             if (found==0)
               {
               for (i=0; i<=maxpid; i=i+1)
                 {
                 j=0;                  /* need to find it in list*/
                 while (typelist[i][j].used==1)
                  {
                  if (strcmp(token_buf,
                       typelist[i][j].sname)==0)  /* a match?  */
                     {
                     strcpy(buffer1,primt[i].nam2); /*yes, save*/
                     found=1;
                     break;              /* and continue */
                     }
                  else
                     j++;  /* no, look at next TYPE */
                  }   /* end while typelist*/
                 if (found==1)
                   break;
                 }    /* end for  */
               }    /* end if found */
             }      /* end else */

            strcpy(swaplist[line_count][line_item].sname,
                                  token_buf);    /* one by one */
            swaplist[line_count][line_item].used=1;

            line_item++;
            } while (delimiter !=5);    /* stop for a ')'   */
```

```
            ckt_line(&outcount,r1,outstack,&incount,instack,&skip1);

            for (i=0; i<outcount-1; i++)          /* write outputs*/
               {
               fprintf(r2," %s ,",outstack[i]);
               if (adflag==1)
                  fprintf(adds," %s ,",outstack[i]);
               }
            fprintf(r2," %s = ",outstack[i]);
            if (adflag==1)
               fprintf(adds," %s = ",outstack[i]);

            fprintf(r2,"%s(",savfunc);  /* write the function name */
            if (adflag==1)
               fprintf(adds,"%s(",savfunc);

            for (i=0; i<incount-1; i++)         /* write the inputs*/
               {
               fprintf(r2," %s ,",instack[i]);
               if (adflag==1)
                  fprintf(adds," %s ,",instack[i]);
               }
            fprintf(r2," %s ) ;\n",instack[i]);
            if (adflag==1)
               fprintf(adds," %s ) ;\n",instack[i]);

            if (expand_flag==1)       /* need to call for expand? */
               {
               for (i=0; i<req_count; i++)      /* yes, see if */
                  {                             /* already did */
                  if (strcmp(reqtable[i].e_from,savfunc)==0)
                     break;                  /* if already marked this  */
                  }                          /* one then ignore it      */
               if (i==req_count)
                  {
                  strcpy(reqtable[i].e_to,buffer1);  /*expand to here */
                  strcpy(reqtable[i].e_from,savfunc); /* from here    */
                  req_count++;
                  expand_flag=0;
                  }
               }

            fprintf(r2," ENDSWAP ; \n");
            line_count++;                     /* increment swaplist */
            getid(r1,33);                     /* then read next token*/
            find_token();
            break;

default:    if (divert==0)
               {
               pdelim(r2);
               if (adflag==1)
                  {
                  count--;
                  pdelim(adds);
                  }
```

```
                    }
                 else
                    pdelim(specs);

                  getid(r1,33);       /* and read the next one     */
                  find_token();
                  break;

          }  /* switch */

       }  /*while*/

    fclose(r1);
    fprintf(adds," END ; \n");       /*       END the file          */
    fclose(adds);
    fprintf(specs," END ; \n");       /*       END the file          */
    fclose(specs);
    print_select=0;                  /* restore to initial state  */
    }
/*-------------------------------------------------------------*/


/*-----------------------uexpand()-----------------------------*/
/* This routine prints any system-generated expansion requests  */
/* onto the file r2 (OUTFILE).                                  */
uexpand()
{
int i;
for (i=0; i<req_count; i++)
   fprintf(r2," EXPAND: %s : %s ;\n",reqtable[i].e_from,reqtable[i].e_to);
}
/*-------------------------------------------------------------*/


/*---------------------- struc_expand() ----------------------*/
struc_expand()

{
int skip,passdown;
modules_to_do=cellcount;
while (expdone==0)
   {
   getid(r1,41);                    /* scan along until found { */
   find_token();
   while (toknn!=4)                   /* looking for { (indicates */
     {                                /* we're in the circuit proper) */
     getid(r1,41);                    /* scan along until found { */
     find_token();
     }

   skip=0;
   while (skip==0)                /* repeat until we find a '}' */
      {
      ckt_line(&outcount,r1,outstack,&incount,instack,&skip);
      if (skip==1)
         {
```

```
            break;                      /* quit        */
            }
         else
            {
            strcpy(token_buf,savfunc);    /* get the function name and*/
            findprim();                   /* see if it is a primitive  */

            if (primid<prim_count)        /* yes, but is it a valid one?*/
               {                          /* if not, let Ausif handle it*/
               if (features[primid][0]==1)   /* STRUC-only description? */
                  {
                  passdown=primid;
                  append(passdown,r2);      /*add it to work file */
                  }  /* if features */
               } /* if primid */
            }  /* else */
         }        /*while skip */

      modules_to_do--;                 /* hey, finished one         */
      cellcount--;                     /* decrement the cell count  */
      if (cellcount==0)                /* cell count =0?            */
         {
         fprintf(r2, "END; \n");       /* yes, place the END on the */
         fclose(r1);                   /* expanded file and close it*/
         fclose(r2);
         if (modules_to_do==0)         /* modulesTodo also = 0?     */
            expdone=1;                 /* yes, then we're finished  */
         else
            {                          /* no, further expansion is  */
            r1=fopen("outfile","r"); /* required--make a new file */
            r2=fopen("infile","w");  /* the old OUTFILE becomes   */
            fcopy(r1,r2);              /* the new INFILE            */
            fclose(r1);
            fclose(r2);

            r1=fopen("infile","r");  /* build the new OUTFILE by  */
            r2=fopen("outfile","w"); /* copying everything but the*/
            copy_noend(r1,r2);         /* END token                 */
            fclose(r1);

            r1=fopen("infile","r");
            countcells(r1);            /* count the # of cells in   */
            fclose(r1);                /* this new input file       */
            r1=fopen("infile","r");  /* now that we've got the files*/
            struc_expand();            /* straight, expand recursively*/
            }
         }   /*  else  */
      }

}
/*-------------------------------------------------------------*/

/*------------------------ append() ------------------------*/
append(ptarget,writefile)
   int ptarget;
   FILE *writefile;
```

```
{
int aindex,done;
FILE *lookfile;
done=0;
aindex=0;
print_select=1;                          /* put linefeeds after ')' */
do
   {
   if (append_table[aindex]==ptarget)    /* already added this one?*/
      break;                             /* yes, skip              */
   aindex++;
   } while (aindex<=append_index);

if (aindex>append_index)                 /* no, append this prim  */
   {
   lookfile=fopen("struc","r");
   do
      {
      getid(lookfile,33);             /* no, wrong primitive*/
      findprim();
      find_token();
      if (toknn==0)                          /* found a MODULE? */
         {                                   /* yes, is it the right one?*/
         getid(lookfile,33);                 /* get the primitive name*/
         findprim();
         if ((primid<prim_count) && (primid==ptarget))
            {
            done=1;                           /* yes, quit          */
            fprintf(writefile,"MODULE : ");
            pdelim(writefile);
            getid(lookfile,33);
            find_token();
            while (toknn!=3)
               {
               pdelim(writefile);
               getid(lookfile,33);    /* write everything down to TYPES */
               find_token();
               }

            pdelim(writefile);        /* write the TYPES token */
            getid(lookfile,33);
            find_token();
            if (toknn==4)             /* no TYPES ? */
               {
               fprintf(writefile, " ; \n");     /* start new line */
               fprintf(writefile, " ( \n");     /* print the (    */
               getid(lookfile,33);              /* then grab token*/
               find_token();
               }

            while (toknn!=5)                    /* copy down to ')'   */
               {
               pdelim(writefile);                /* copy the token     */
               getid(lookfile,33);               /* and get the next one */
               find_token();
               }
```

```
                pdelim(writefile);                    /*write the ')'       */
                append_table[append_index]=primid;   /*and update the table */
                modules_to_do++;          /* I hate recursion              */
                append_index++;
                }
            }
          } while (done==0);

       fclose(lookfile);
       } /* if */
    print_select=0;                       /* restore to previous state */
    }    /* append */
/*-------------------------------------------------------*/


/*-------------------------swap()-------------------------*/
/* This routine is invoked after expansion and just before*/
/* compilation.  If there are any requests for function   */
/* name substitution with USING, they will be handled     */
/* here.  The affected lines are delimited by SWAPLIN and */
/* ENDSWAP keywords which will be removed prior to compil-*/
/* ation.                                                 */
swap()
{
int skip1, found, i, j;
int perform, swaps;

ri=fopen("p11","r");                  /* raw file is P11 */
wi=fopen("outfile","w");              /* processed file goes here */

getid(ri,41);
find_token();
while (toknn!=4)
   {
   pdelim(wi);                          /* copy down to the '('  */
   getid(ri,41);
   find_token();
   }

pdelim(wi);                             /* write the  '('         */
swaps = -1;
found_start=0;
found_end=0;

skip1=0;
while (1)
   {
   ckt_line(&outcount,ri,outstack,&incount,instack,&skip1);
   if (skip1==1)                      /* quit when we see a ')'  */
      break;
   if (found_end==1)                  /* find an ENDSWAP while getting ckt line?*
      {
      perform=0;                      /* no need to check for swaps */
      found_end=0;                    /* set up for the next ENDSWAP key */
      }
   if (found_start==1)                /* find a SWAPLIN keyword?      */
```

```
    {
    perform=1;                      /* need to start looking for swaps */
    found_start=0;                  /* set up for next SWAPLIN key  */
    swaps++;                        /* select the next line of swaplist */
    }

if (perform==1)                     /* check for swaps on this ckt line? */
    {
    strcpy(token_buf,savfunc);      /* move function name to token buffer*/
    findprim();                     /* and see what type it is           */
    i=0;                            /* initialize USING param selector   */
    if (typelist[primid][0].used==1)    /* this TYPE is user-defined?  */
       {
       found=0;                     /* yes, initialize to "not found"  */

       while (swaplist[swaps][i].used==1)    /* match USING param list..*/
          {
          j=0;

          while (typelist[primid][j].used==1)  /*..against TYPE list  */
             {
             if (strcmp(swaplist[swaps][i].sname,
                          typelist[primid][j].sname)==0)  /* a match?  */
                {
                strcpy(savfunc,typelist[primid][j].sname);  /* yes, swap */
                found=1;                 /* tell outside world we're done */
                break;                   /* and quit           */
                }
             else
                j++;                     /* no, look at next TYPE */
             }            /* while typelist*/

          if (found==1)                  /* are we done? */
             break;                      /* yes, look for next substitution */
          else
             i++;                        /* no, try next USING parameter */
          }   /* while swaplist */

       if (found==0)
          {
          printf("couldn't find the module you wanted to replace %s\n",
                                    savfunc);
          printf("compilation continues\n\n");
          }

       }  /* if typelist */
    }   /* if perform */

    for (i=0; i<outcount-1; i++)                  /* write outputs*/
       fprintf(wi," %s ,",outstack[i]);
    fprintf(wi," %s = ",outstack[i]);

    fprintf(wi,"%s(",savfunc);          /* write the function name */

    for (i=0; i<incount-1; i++)                /* write the inputs*/
       fprintf(wi," %s ,",instack[i]);
```

```
      fprintf(w1," %s ) ;\n",instack[i]);

   }  /* while not skip  */

pdelim(w1);                              /* write the ')'          */
fcopy(r1,w1);                            /* then copy rest of file*/
fclose(r1);
fclose(w1);
r1=fopen("outfile","r");
r2=fopen("specs","r");
w1=fopen("p11","w");

print_select=1;
getid(r1,33);
find_token();
while (toknn!=5)
   {
   pdelim(w1);               /* copy circuit body */
   getid(r1,33);
   find_token();
   }

pdelim(w1);
print_select=0;
getid(r2,33);
find_token();
if (toknn==20)
   fprintf(w1,"DEFINE: ;\n");
else
   {
   while (toknn!=20)
      {
      pdelim(w1);              /* copy the DEFINEd parameters */
      getid(r2,33);
      find_token();
      }
   }
while (toknn!=6)             /* look for the INITIAL:   */
   {
   getid(r1,33);
   find_token();
   }
pdelim(w1);
fclose(r2);
fcopy(r1,w1);                            /* copy the processed file */
fclose(r1);                              /* back to p11             */
fclose(w1);
}
/*-------------------------------------------------------------*/


/*----------------------check_deeper()-------------------------*/
check_deeper(fnam,targetoid,targetname)
char fnam[];
int targetoid;
char targetname[];
```

```
{
FILE *lib;
int findex,skip;
int incnt,outcnt, i, intable;
char instk[maxouts][8], ostk[maxouts][8] ;
struct functable ftable[maxprim];

lib=fopen("struc","r");
getid(lib,33);
find_token();
while (1)
   {
   if (toknn==0)                    /* find a MODULE? */
      {
      getid(lib,41);                /* yes, is it the right one? */
      if (strcmp(fnam,token_buf)==0)
         {
         while (toknn!=4)           /* yes, read down to circuit body */
            {
            getid(lib,41);
            find_token();
            }
         break;
         }
      }

   getid(lib,44);
   find_token();
   }

findex=0;
skip=0;
while(1)
   {
   ckt_line(&outcnt,lib,ostk,&incnt,instk,&skip);
   if (skip==1)
      break;
   strcpy(ftable[findex].fnname,savfunc);
   strcpy(token_buf,savfunc);
   findprim();
   ftable[findex].level=features[primid][1];
   findex++;
   }

fclose(lib);

for (i=0; i<findex; i++)
   {
   checktable(i, &intable, ftable);          /* this one in expand table? */
   if ((ftable[i].level > features[targetpid][1]) && (intable==0))
      {
      strcpy(expt[expcount].fname,ftable[i].fnname); /* add it to table */
      expcount++;
      check_deeper(ftable[i].fnname,targetpid,targetname);
      }
   else
```

```
      {
      if ((ftable[i].level==features[targetoid][i]) &&
                  (strcmp(ftable[i].fnname,targetname)==0))
         {           /* function name what we're looking for? */
         sfound=1;       /* yes, found at least one occurrance */
         }
      }
   }
}
/*-------------------------------------------------------------*/


/*---------------------checktable()----------------------------*/
checktable(index,result,table)
int index,*result;
struct functable table[];
{
int i;

*result=0;                    /* initialize to "not found"  */
for (i=0; i<expcount; i++)
   {
   if (strcmp(expt[i],fname,table[index].fnname)==0)  /* find name in extbl? */
      {
      *result=1;
      break;
      }
   }
}
/*-------------------------------------------------------------*/


/*--------------------perform_expansion()----------------------*/
perform_expansion()
{
extern int occurance;
int i,k,edone,end;

for (k = 0; k < expcount; k = k + 1)  /* each expansion request*/
   {                                  /* handled one at a time */
   r1 = fopen("p11","r");
   w1 = fopen("p1exp","w") ;/* declaration part of expanded*/
                                 /* user program                */

   secondp(k) ;                 /* second pass - expansion     */
   fclose(r1) ;                 /* expanded desc. is in SCR2   */
                                 /* expanded declaration inP1EXP*/
   /*---------------copy P1EXP to temp-----------------------*/
   t1 = fopen("temp","w") ;
   r1 = fopen("p1exp","r") ;
   fcopy(r1,t1) ;
   fclose(r1) ;
   /*-------------------------------------------------------*/

   /*-----------copy internals of function to t1 with tacking----*/
   s1 = fopen("scr1","r") ;   /* function's description    */
```

```
   while (1)    /* read internals and copy to temp with tacking */
      {
      getid(s1,41);        /* 41 = missing { error */
      find_token() ;

      if (toknn == 4)      /* { */
         break ;
      if (toknn != 8)      /* INTERNAL */
         {
         end = 0 ;
         for (i = 0; i < occurance; i = i + 1)
            {
            tack(i, token_buf) ;
            if (delimiter == 2)    /* ';'   */
               {
               end = 1 ;
               if ((i + 1) != occurance)
                  delimiter = 1 ;
               }
            pdelim(t1) ;
            if (end == 1)
            delimiter = 2 ;
            }
         }
      else
         pdelim(t1) ;
      }                     /* end while */

   fprintf(t1," {  \n") ;
   fclose(s1) ;
   /*-------------------------------------------------------*/

   /*------------append SCR2 to temp and copy back to P11--*/
   s2 = fopen("scr2","r") ;
   fcopy(s2,t1) ;
   fprintf(t1," }\n") ;
   fclose(s2) ;
   fclose(t1) ;

   t1 = fopen("temp","r") ;
   w1 = fopen("p11","w") ;
   fcopy(t1,w1) ;
   fclose(t1) ;
   fclose(w1) ;
   /*-------------------------------------------------------*/
   }  /* end for */

if (expcount == 0)
   printf(" >>> No expansion request \n");
else   /* tack last part (simulation control specs) to P11 */
   {
   r1=fopen("p11","r");
   t1 = fopen("temp","w") ;
   fcopy(r1,t1);
   fclose(r1);
   r1 = fopen(userprg,"r") ;/* user program   */
```

```
getid(r1,33);
find_token();
while (toknn!=5)
   {
   getid(r1,33);
   find_token();
   }
edone=0;
getid(r1,33);                  /* get next token after ) */
find_token();
while (1)
   {
   switch(toknn)
      {
      case 0:  edone=1;            /* MODULE */
               break;

      case 6:  print_select=1;
               pdelim(t1);
               getid(r1,34);
               find_token();
               if (toknn==7)
                  fprintf(t1," ; \n");
               break;

      case 9:  print_select=0;
               pdelim(t1);
               getid(r1,34);
               find_token();
               if (toknn==6)
                  {
                  fprintf(t1," ; \n");
                  pdelim(t1);
                  getid(r1,34);
                  find_token();
                  if (toknn==7)
                     fprintf(t1," ; \n");
                  }
               break;

      case 20: edone=1;            /* END */
               break;

      case 21: getid(r1,34);       /* EXPAND */
               getid(r1,34);
               find_token();
               break;

      default: pdelim(t1);
               getid(r1,34);
               find_token();
               break;

      }   /* switch*/
```

```
        if (edone==1)
           break;
        }       /* while */

    fclose(t1) ;

    t1 = fopen("temp","r") ;
    w1 = fopen("p11","w") ;
    fcopy(t1,w1) ;
    fclose(t1) ;
    fclose(w1) ;
    }
 }
 /*------------------end expansion-----------------------------*/


/*------------------------add_lib()-----------------------------*/
add_lib()
{
int i,icnt,ocnt,done;
int skip,maxlevel;

done=0;
print_select=1;
r1=fopen("newckts.vhl","r");
getid(r1,33);                              /* find out which addition to do */
find_token();
do
   {
   icnt=0;
   ocnt=0;
   maxlevel=0;

   switch (toknn)
      {
      case 20: done=1;               /* found the END token */
               break;

      case 26: printf("ADDCELL not installed yet....\n");
               break;

      case 27: printf("Performing a structure-only addition for.. ");
               r2=fopen("temp","w");
               r3=fopen("struc","r");
               fcopy(r3,r2);
               fclose(r3);
               getid(r1,33);
               pdelim(r2);                        /* write MODULE token */
               getid(r1,33);                      /* get the module name */
               strcpy(prim[sys_prims],nam2,token_buf);
               printf("%s\n",token_buf);
               pdelim(r2);
               getid(r1,33);                      /* get INPUTS keyword */
               pdelim(r2);
               getid(r1,33);                      /* get first input name */
               find_token();
```

```
while (toknn!=2)                    /* stop when find OUTPUTS */
   {
   icnt++;                          /* count and write inputs */
   pdelim(r2);
   getid(r1,33);
   find_token();
   }
primt[sys_prims].numoar=icnt;
pdelim(r2);                          /* write OUTPUTS keyword */
getid(r1,33);
find_token();                       /* get first output  */
while (toknn!=3)                    /* stop when find TYPES */
   {
   ocnt++;                          /* count and write outputs */
   pdelim(r2);
   getid(r1,33);
   find_token();
   }
primt[sys_prims].outp=ocnt;
features[sys_prims][0]=1;

pdelim(r2);                          /* write the TYPES token   */
getid(r1,33);                       /* get the next token      */
find_token();
if (toknn==4)                       /* this will happen if no TYPES*/
   {                                /* are declared            */
   count=0;
   fprintf(r2," ; \n");             /* puts TYPES: ; into file */
   pdelim(r2);                      /* then print the '(' */
   }
else
   {
   while (toknn!=4)
      {                             /* write all the TYPES */
      pdelim(r2);
      getid(r1,33);
      find_token();
      }
   pdelim(r2);                      /* then write the ')'  */
   }

skip=0;

while (1)                  /* get the rest of the circuit*/
   {
   ckt_line(&outcount,r1,outstack,&incount,instack,&skip);
   if (skip==1)
      break;
   for (i=0; i<outcount-1; i++)          /* write outputs*/
      fprintf(r2," %s ,",outstack[i]);
   fprintf(r2," %s = ",outstack[i]);

   fprintf(r2,"%s(",savfunc);  /* write the function name */

   for (i=0; i<incount-1; i++)         /* write the inputs*/
      fprintf(r2," %s ,",instack[i]);
```

```
                fprintf(r2," %s ) :\n",instack[i]);

                strcpy(token_buf,savfunc);
                findprim();                          /*see what func name is*/
                if (features[primid][1] > maxlevel)  /*if higher level */
                   maxlevel=features[primid][1];  /*then save higher level*/
                }

             fprintf(r2," } \n");            /* write the closing brace */
             fclose(r2);

             features[sys_prims][1]=maxlevel+1; /*this prim is 1 level*/
             sys_prims++;                /* higher than highest subckt*/

             r2=fopen("temp","r");
             if (r2==NULL)
                printf("Temp file open failed\n");
             r3=fopen("struc","w");
             if (r3==NULL)
                printf("Struc file open failed\n");
             fcopy(r2,r3);                    /* copy new file back to */
             fclose(r2);                      /* STRUC                 */
             fclose(r3);
             printf("writing PRIMITIV.DAT file\n");
             r4=fopen("primitiv.dat","w");
             if (r4==NULL)
                printf("Primitive file open failed\n");
             fprintf(r4,"%d \n",sys_prims);
             for (i=0; i<sys_prims; i++)
                {
                fprintf(r4," %s %d %d %d %d\n",prim[i].nam2,prim[i].number,
                      prim[i].outp,features[i][0],features[i][1]);
                }
             fclose(r4);
             getid(r1,33);
             find_token();
             break;

      case 28: printf("ADDBLOCK not installed yet....\n");
               break;

      } /*switch */

   } while (done==0);
print_select=0;
}/*addlib*/
```

```
/*****************************************
      Logic Primitive Descriptions File
*****************************************/
#include "stdio.h"

#define maxprim 100

struct prim_tab {
     char nam[8],nam2[8];
     int numpar,outp;
     int normld, fanout;
     int technology, overld;
     };
struct prim_tab *tempptr;

primsetup(primptr)
   struct prim_tab *primptr;
   {
   extern int prim_count,features[maxprim][2];
   int i;
   char ch;
   FILE *xp;

  tempptr=primptr;
  strcpy((*primptr).nam, "READIN") ;
  primptr++;
  strcpy((*primptr).nam, "AND") ;
  primptr++;
  strcpy((*primptr).nam, "OR") ;
  primptr++;
  strcpy((*primptr).nam, "NAND") ;
  primptr++;
  strcpy((*primptr).nam, "NOR") ;
  primptr++;
  strcpy((*primptr).nam, "INVERT") ;
  primptr++;
  strcpy((*primptr).nam, "EXOR") ;
  primptr++;
  strcpy((*primptr).nam, "ANDTHRE") ;
  primptr++;
  strcpy((*primptr).nam, "NANDTHR") ;
  primptr++;
  strcpy((*primptr).nam, "SRBLOCK") ;
  primptr++;
  strcpy((*primptr).nam, "RETDBLO") ;
  primptr++;
  strcpy((*primptr).nam, "ANDFOUR") ;
  primptr++;
  strcpy((*primptr).nam, "NANDFOU") ;
  primptr++;
  strcpy((*primptr).nam, "ORTHREE") ;
  primptr++;
  strcpy((*primptr).nam, "ORFOUR") ;
  primptr++;

  strcpy((*primptr).nam,"HALFADD");       /* set up user-defined prims */
```

```
primptr++;
strcpy((*primptr).nam,"USER2");        /* we'll check to see if any */
primptr++;
strcpy((*primptr).nam,"USER3");       /* are actually present later */
primptr++;
strcpy((*primptr).nam,"USER4");
primptr++;
strcpy((*primptr).nam,"USER5");
primptr++;
strcpy((*primptr).nam,"USER6");
primptr++;
strcpy((*primptr).nam,"USER7");
primptr++;
strcpy((*primptr).nam,"USER8");
primptr++;
strcpy((*primptr).nam,"USER9");
primptr++;
strcpy((*primptr).nam,"USER10");
primptr++;
strcpy((*primptr).nam,"USER11");
primptr++;
strcpy((*primptr).nam,"USER10");
primptr++;
strcpy((*primptr).nam,"USER12");
primptr++;
strcpy((*primptr).nam,"USER13");
primptr++;
strcpy((*primptr).nam,"USER14");
primptr++;
strcpy((*primptr).nam,"USER15");
primptr++;
strcpy((*primptr).nam,"USER16");
primptr++;
strcpy((*primptr).nam,"USER17");
primptr++;
strcpy((*primptr).nam,"USER18");
primptr++;
strcpy((*primptr).nam,"USER19");
primptr++;
strcpy((*primptr).nam,"USER20");
primptr++;
strcpy((*primptr).nam,"USER21");
primptr++;
strcpy((*primptr).nam,"USER22");
primptr++;
strcpy((*primptr).nam,"USER23");
primptr++;
strcpy((*primptr).nam,"USER24");
primptr++;
strcpy((*primptr).nam,"USER25");
primptr++;
strcpy((*primptr).nam,"USER26");
primptr++;
strcpy((*primptr).nam,"USER27");
primptr++;
strcpy((*primptr).nam,"USER28");
```

```
    primptr++;
    strcpy((*primptr).nam,"USER29");
    primptr++;
    strcpy((*primptr).nam,"USER30");
    primptr++;
    strcpy((*primptr).nam,"USER31");
    primptr++;
    strcpy((*primptr).nam,"USER30");
    primptr++;
    strcpy((*primptr).nam,"USER32");
    primptr++;
    strcpy((*primptr).nam,"USER33");
    primptr++;
    strcpy((*primptr).nam,"USER34");
    primptr++;
    strcpy((*primptr).nam,"USER35");
    primptr++;
    strcpy((*primptr).nam,"USER36");
    primptr++;
    strcpy((*primptr).nam,"USER37");
    primptr++;
    strcpy((*primptr).nam,"USER38");
    primptr++;
    strcpy((*primptr).nam,"USER39");
    primptr++;
    strcpy((*primptr).nam,"USER40");

    primptr=tempptr;                    /* reset pointer to start */

    xp=fopen("primitiv.dat","r");     /*  get additional info from disk*/
    fscanf(xp,"%d",&prim_count);
    for (i=0; i<prim_count; i++)
       {
       fscanf(xp,"%s %d %d %d %d",(*primptr).nam2,&(*primptr).number,
                      &(*primptr).outp,&features[i][0],&features[i][1]);
       primptr++;
       }
    fclose(xp);

    }
/*-------------------------------------------------------------*/
```

APPENDIX C.

VOHL COMPILER DATA FILES

STRUC	The primitive library containing VOHL structural descriptions; the expanded primitive descriptions.
PRIMITIV.DAT	The new data file containing the user's names for the primitives and various other information.
BLDEL	The block delays for each primitive; that is, the propagation delay from each input to each output.

```
MODULE : INVERT ;
INPUTS : A ;
OUTPUTS : B ;
TYPES: ;
{
0:
}
MODULE : AND ;
INPUTS : A, B ;
OUTPUTS : Z ;
TYPES: ;
{
0;
}
MODULE : OR ;
INPUTS : A, B ;
OUTPUTS : Z ;
TYPES : ;
{
0:
}
MODULE : ANDTHREE ;
INPUTS : A, B, C ;
OUTPUTS : Z ;
TYPES : ;
{
0;
}
MODULE: NANDTHRE ;
INPUTS: A, B, C ;
OUTPUTS : Z ;
TYPES : ;
{
0;
}
MODULE : ORTHREE ;
INPUTS : A, B, C ;
OUTPUTS : Z ;
TYPES : ;
{
0 ;
}
MODULE : EXOR ;
INPUTS : A, B ;
OUTPUTS : Z ;
TYPES : ;
{
0 ;
}
MODULE : NAND  ;
INPUTS : A, B ;
OUTPUTS : Z ;
TYPES : ;
{
0 ;
}
```

```
MODULE : NOR ;
INPUTS : A, B ;
OUTPUTS : Z ;
TYPES : ;
{
0 ;
}
MODULE : ANDFOUR ;
INPUTS : A, B, C, D ;
OUTPUTS : Z ;
TYPES : ;
{
0 ;
}
MODULE : NANDFOUR ;
INPUTS : A, B, C, D ;
OUTPUTS : Z ;
TYPES : ;
{
0 ;
}
MODULE : ORFOUR ;
INPUTS : A, B, C, D ;
OUTPUTS : Z ;
TYPES : ;
{
0 ;
}
MODULE : SRBLOCK ;
INPUTS : S, R, X ;
OUTPUTS : Q, QC, Y ;
TYPES : ;
{
 Q = NAND(S, QC) ;
 QC = NAND(Q, R) ;
}
MODULE : RETDBLO ;
INPUTS : CLR, D, CLK, DUM1, DUM2, DUM3 ;
OUTPUTS : Q, QC, DM1, DM2, DM3 ;
TYPES : INTERNALS : X, Y, W, Z ;
{
 X = NAND(Z, Y) ;
 Y = NANDTHRE( X,CLR, CLK) ;
 W = NANDTHRE(Y, CLK, Z) ;
 Z = NANDTHRE(W, CLR, D) ;
 Q = NAND(Y, QC) ;
 QC = (NANDTHRE(Q, CLR, W) ;
}
 MODULE : JKFF ;
 INPUTS : J , K , CLK ;
 OUTPUTS : Q , QBAR ;
 TYPES : INTERNA : S1 , R1 , D1 , D2 ;
 {
 S1 = NANDTHR ( QBAR , J , CLK ) ;
 R1 = NANDTHR ( CLK , K , Q ) ;
 Q , QBAR , D1 = SRBLOCK ( S1 , R1 , D2 ) ;
```

```
}
MODULE : FULLADD ;
INPUTS : A , B , CIN ;
OUTPUTS : S , CO ;
TYPES : INTERNA : X , Y , Z ;
{
X , Y = HALFADD ( A , B ) ;
S , Z = HALFADD ( X , CIN ) ;
CO = OR ( Z , Y ) ;
}
MODULE : HALFADD ;
INPUTS : C , D ;
OUTPUTS : T , COH ;
TYPES : ;
{
T = EXOR ( C , D ) ;
COH = AND ( C , D ) ;
}
```

```
15
READIN     0  0  2  0
AND        2  1  2  0
OR         2  1  2  0
NAND       2  1  2  0
NOR        2  1  2  0
INVERT     1  1  2  0
EXOR       2  1  2  0
ANDTHRE    3  1  2  0
NANDTHR    3  1  2  0
SRBLOCK    3  3  2  1
RETDBLO    6  5  2  1
ANDFOUR    4  1  2  0
NANDFOU    4  1  2  0
ORTHREE    3  1  2  0
ORFOUR     4  1  2  0
```

The Data File **PRIMITIV.DAT**

```
AND
2
0  0  1  1
1  0  1  1
OR
2
0  0  1  1
1  0  1  1
NAND
2
0  0  1  1
1  0  1  1
NOR
2
0  0  1  1
1  0  1  1
INVERT
1
0  0  1  1
ANDTHRE
3
0  0  1  1
1  0  1  1
2  0  1  1
NANDTHR
3
0  0  1  1
1  0  1  1
2  0  1  1
SRBLOCK
8
0  0  2  2
2  0  2  2
0  1  2  2
2  1  2  2
1  1  2  2
1  2  2  2
0  2  2  2
2  2  2  2
RETDBLO
24
0  0  2  2
2  0  2  2
3  0  2  2
4  0  2  2
5  0  2  2
0  1  2  2
2  1  2  2
3  1  2  2
4  1  2  2
```

```
5  1  2  2
0  2  2  2
1  2  2  2
2  2  2  2
3  2  2  2
4  2  2  2
0  3  2  2
2  3  2  2
3  3  2  2
4  3  2  2
0  4  2  2
2  4  2  2
3  4  2  2
4  4  2  2
5  4  2  2
ANDFOUR
4
0  0  1  1
1  0  1  1
2  0  1  1
3  0  1  1
NANDFOU
4
0  0  1  1
1  0  1  1
2  0  1  1
3  0  1  1
ORTHREE
3
0  0  1  1
1  0  1  1
2  0  1  1
ORFOUR
4
0  0  1  1
1  0  1  1
2  0  1  1
3  0  1  1
EXOR
2
0  0  1  1
1  0  1  1
END
```

APPENDIX D.

SIMULATOR SOURCE CODE

TWHEEL.C The timing wheel main module.

```
/*****************************************************************
 *                                                               *
 *                      Timing Wheel Program                     *
 *                            for the                            *
 *                   Multilevel Logic Simulator                  *
 *                                                               *
 *                      Version 1.1  25 Aug 86                   *
 *     Original Version by Ausif Mahmood at WSU for UNIX VAX     *
 *   Microcomputer versions and bug fixes by Scott Kelly at NPS  *
 *****************************************************************/

int _mlen = 250 ;
int _mem[250] ;
#include "stdio.h"

/*
 This is the timing wheel program. First it initializes the circuit
 inter-connections in terms of descriptors (descriptor interconnec-
 tions are given in a file  "p2a"). The circuit is then  simulated
 according to the input data given in a file. Event directed simula-
 tion is used for maximum time efficiency -Ulrich's algorithm.
 Simulation results are directed to a file .  */


#define inputs 2            /*  2 inputs per descriptor allowed   */
#define maxprim 100         /*  maximum number of primitives      */

#define maxdescrpt 250     /*  maximum number of descriptors     */

#define maxsymb 50         /*  symbol table for printing         */

#define mdelay 100          /*  maximum possible delay            */

#define mdepth 250          /*  concurrent actions                */

#define maxmat 400          /*  delay matrix units                */

#define maxinput 100        /*  max number of inputs              */

#define maxtw 400           /*  timing wheel data structure       */

#define maxout 32           /*  maximum number of outputs per prim*/



/*---------------------------------------------------------------*/

struct matrix {             /* delay matrix for a primitive       */
  int rdelay0, rdelay1, fdelay0, fdelay1 ;
  struct matrix *matptr ;
 } ;
/*---------------------------------------------------------------*/
```

```
/*--------------RECORD ORGANIZATION FOR A DESCRIPTOR-------------*/
struct descrpt {


  int (*pfunc)() ;          /*  pfunc is pointer to the function  */

                            /*  i.e. code for the primitive       */

  int param[7] ;            /*  parameters for the function       */

                            /*  param[0] = input1 value           */

                            /*  param[1] = input2 value           */
                            /*  param[2] = rise1 delay            */
                            /*  param[3] = fall1 delay            */

                            /*  param[4] = rise2 delay            */

                            /*  param[5] = fall2 delay            */

                            /*  param[6] = MODE ( 0 = normal )    */

                            /*        ( 1 = uncertain if low )    */

                            /*        ( 2 = uncertain if high)    */

                            /*        ( 3 = stuck-at-0 fault)     */

                            /*        ( 4 = stuck-at-1 fault)     */


  struct matrix *mptr ;     /* mptr is pointer to delay matrix    */


  struct descrpt *header ; /* header is a pointer to a descriptor*/

  struct descrpt *right0 ; /* 2 right pointers per descriptor    */
  struct descrpt *right1 ; /* block.                             */

  int h_value  ;            /* field indicator for header pointer */

  int r_value[inputs] ;     /* field indicators for right pointers*/


  int present_output  ;     /* present output of a primitive      */


  struct descrpt *ext_ptr ;/* extension pointer for multi-        */

                            /* descriptor primitives.            */
```

```
  int parent ;              /* parent descriptor number          */

 } ;

/*-------------------------------------------------------------*/



/*---------------------TIMING WHEEL STRUCTURES------------------*/

 struct newstack {          /* new_values for multi-output       */

   int newvalue ;           /* functions attached to timing wheel */

   struct newstack *newptr;

 } ;


 struct time_stack {        /* timing_wheel structure            */
   struct descrpt *dptr ;   /* dptr = pointer to the descriptor  */
   int newval, sflag ;      /* newval= newvalue, sflag= scheduling*/
   struct newstack *nptr ;  /* flag, nptr = pointer to newstack  */

   struct time_stack *tptr ;

 } ;

/*-------------------------------------------------------------*/



  FILE *rp           ;      /*  Read pointer to input data file  */

  FILE *wp           ;      /*  Write pointer to output data file */



/*----------------SYMBOL TABLE RECORD STRUCTURE-----------------*/

  struct symb_tab {
    char name[8]  ;         /* name of the line                  */
    int index     ;         /* index = # of the associated desc. */
                  } ;
/*-------------------------------------------------------------*/


/*----------------------GLOBAL PARAMETERS----------------------*/
  struct descrpt desc[maxdescrpt]; /*   STORAGE FOR STRUCTURES   */
```

```
  struct time_stack tstack[maxtw] ;

  struct newstack nstack[mdepth] ; /* timing wheel structures     */


  struct matrix matx[maxmat] ;


  struct symb_tab svm[maxsymb] ;



  struct descrpt *ptr, *parent_ptr ;


  struct newstack *nwptr, *fref, *nptr, *nwipt ;

  struct time_stack *endt[mdelay], *begint[mdelay], *freft ;
  struct time_stack *savt, *fwdt, *tempt ;

  int num_outs ;                  /* # of outputs for a function  */

  int time, sav_time, sim_time ;


  int sav_write ;          /* write flag for printing output names*/


  int f_out[maxout] ;      /* outputs returned by each function   */

  int ff_out[maxout] ;

  int del[maxout], delay ;


  int timing_wheel, depth[mdelay] ;
/* depth[]  indicates the number of concurrent actions in 1 slot */

  int num_input, inpt[maxinput] , inptc[maxinput] ;

  int hashtable[100];               /* simple hashtable for variable names */
  int hashcount;                    /* number of items in hashtable        */


/*  num_input = number of inputs in the input data file          */

/*  inpt[] = array holding values of inputs                      */
```

```
/*  inptc[]= array holding hash value      of inputs                */


  int num_print , out_interval ;

  char tokenbuff[8] ;

  int endinputname ;

/* num_print  = number of lines to be printed out                   */

/* out_interval=interval after which each output is to be printed*/


  int (*pnfn[maxprim])() ; /* pointer to primitive functions      */
  int pncnt ;

  int AND() ;
  int OR() ;
  int INVERT() ;
#include "ftype"

  int fmode() ;             /* mode behavior simulation */
  int fdel_ins() ;          /* finds delays applicable to a change*/
  int insert() ;            /* insert the desc. in proper slot    */
  int indata() ;            /* input data retrieval for descs.    */

  int read_input() ;
  int write_output() ;
  int hashf() ;
  int test();
  int getname() ;
/*-----------------------------------------------------------------*/


/*----------------------------MAIN PROGRAM-------------------------*/
main(argc,argv)
int argc ;
char *argv[] ;
 {

 int i, j, k, fx, field_no, duminp[32] ;

 int flag, sav_value , savh;

 int sav_depth, endread, num1, num2, num3, num4 ;
 FILE *wq ;
 struct descrpt *sav_ptr, *prev_ptr ;

 struct matrix *ptrm ;

 char z ;
/*-----------------------------------------------------------------*/
```

```
/*-------------------------BEGIN-----------------------------------*/

hashcount=0;
for (i=0; i<100; i++)
   hashtable[i] = -1;

 rp = fopen(argv[1],"r") ; /*   is the input data file      */

 pnfn[0] = read_input ;
 pnfn[1] = AND ;
 pnfn[2] = OR ;
 pnfn[5] = INVERT ;
#include "faddr"

 printf(" \n");
 printf("    ------------------------------------ \n");
 printf("   | MULTI-SIM version 1.0 -Nov. 1, 1984 |\n");
 printf("   ----------------------------------------\n");
 printf(" \n");
 printf("              Loading Compiler Data...\n");
 printf(" \n");

/*-----------------------INITIALIZE DELAY MATRIX-----------------*/
 for (i = 0; i < maxmat; i = i + 1)
  {
   matx[i].rdelay0 = -1 ;
   matx[i].fdelay0 = -1 ;
   matx[i].rdelay1 = -1 ;
   matx[i].fdelay1 = -1 ;
   matx[i].matptr = NULL ;
  }
/*---------------------------------------------------------------*/

/*---------------------INITIALIZE DESCRIPTORS--------------------*/
 for (i = 0; i < maxdescrpt ; i = i + 1 )
  {
   for (j = 2; j < 6 ;   j = j + 1 )
     desc[i].param[j] = -1 ;       /* initialize parameters to -1*/
   for (j = 0; j < inputs ; j = j + 1 )
     desc[i].r_value[j] = 0 ;
   desc[i].param[0] = 2 ;
   desc[i].param[1] = 2 ;          /* initialize inputs to 2's   */
   desc[i].param[6] = 0 ;          /* normal mode of operation   */
   desc[i].header = &(desc[i]) ;
   desc[i].h_value = 0 ;
   desc[i].parent = i ;
   desc[i].present_output = 2 ;    /*  2 stands for don't care   */
   desc[i].ext_ptr = NULL ;
   desc[i].mptr = NULL ;
  }
/*---------------------------------------------------------------*/

/*--------------------Read input line names----------------------*/
 endinputname = 0 ;
 j = 0 ;
```

```
 while(1)
  {
   getname() ;
   inptc[j] = hasnf(tokenbuff);
   j = j + 1 ;
   if (endinputname == 1)
     break ;
  }
 num_input = j ;
/*-------------------------------------------------------------*/

/*---------INITIALIZE TIMING WHEEL & STORAGE POOL---------------*/
 for ( i = 0; i < mdelay ; i = i + 1 )
  {
   depth[i] = -1;
   begint[i] = NULL ;
   endt[i] = NULL ;
  }


 j = maxtw - 1 ;
 for ( i = 0; i < j ; i = i + 1)
  {
   tstack[i].newval = 2 ;
   tstack[i].sflag  = 2 ;
   tstack[i].tptr = &(tstack[i+1]) ;
   tstack[i].nptr = NULL ;
  }

 freft = &(tstack[0]) ;
/*-------------------------------------------------------------*/

/*-------STORAGE POOL FOR NSTACK(future multi-output values)-----*/
 j = mdepth - 1 ;
 for ( i = 0; i < j; i = i + 1)
   nstack[i].newptr  = &(nstack[i + 1])  ;
 fref = &(nstack[0]) ;  /* fref points to the first free nstack  */
/*-------------------------------------------------------------*/
/*-----------------INITIALIZE CIRCUIT CONNECTIONS--------------*/
  wq = fopen("simdata","r") ;
  endread=0;
  while(endread==0)
   {
    fscanf(wq,"%d",&num1) ;
    switch (num1)
     {
      case 1 : fscanf(wq,"%d %d %d",&num2, &num3, &num4) ;
               switch (num3)
                {
                 case 8 : desc[num2].header = &(desc[num4]);
                          break ;
                 case 9 : if (num4 == 0)
                            desc[num2].right0 = ptr ;
                          else
                            desc[num2].right1 = ptr ;
                          break ;
```

```
            case 11 :desc[num2].h_value = num4 ;
                     break ;
            case 12: desc[num2].r_value[num4] = savh ;
                     break ;
            case 15: desc[num2].ext_ptr = &(desc[num4]) ;
                     break ;
            case 16: desc[num2].parent = num4 ;
                     break ;
            default: if (num3 < 7)
                       desc[num2].param[num3] = num4 ;
                     else
                      {
                       printf(" error in decoding code\n") ;
                        endread = 1 ;
                       break ;
                      }
           }
          break ;
case 2 : fscanf(wq,"%d",&num2) ;
          savh = desc[num2].h_value ;
          break ;
case 3 : fscanf(wq,"%d",&num2) ;
          ptr = desc[num2].header ;
          break ;
case 4 : fscanf(wq,"%d",&num2) ;
          ptr = &(desc[num2]) ;
          break ;
case 5 : fscanf(wq,"%d %d",&num2, &num3) ;
          (*ptr).param[num2] = num3 ;
          break ;
case 6 : fscanf(wq,"%d",&num2) ;
          ptr = desc[num2].ext_ptr ;
          break ;
case 7 : ptr = (*ptr).ext_ptr ;
          break ;
case 8 : ptrm = (*ptr).motr ;
          break ;
case 9 : ptrm = (*ptrm).matptr ;
          break ;
case 10: fscanf(wq,"%d",&num2) ;
          (*ptrm).rdelay0 = num2 ;
          break ;
case 11: fscanf(wq,"%d",&num2) ;
          (*ptrm).fdelay0 = num2 ;
          break ;
case 12: fscanf(wq,"%d",&num2) ;
          (*ptrm).rdelay1 = num2 ;
          break ;
case 13: fscanf(wq,"%d",&num2) ;
          (*ptrm).fdelay1 = num2 ;
          break ;
case 14: fscanf(wq,"%d",&num2) ;
          (*ptr).motr = &(matx[num2]) ;
          break ;
case 15: fscanf(wq,"%d %d",&num2, &num3) ;
          matx[num2].matptr = &(matx[num3]) ;
```

```
            break ;
    case 16: fscanf(wq,"%d %d",&num2, &num3) ;
            matx[num2].rdelay0 = num3 ;
            break ;
    case 17: fscanf(wq,"%d %d",&num2, &num3) ;
            matx[num2].fdelay0 = num3 ;
            break ;
    case 18: fscanf(wq,"%d %d",&num2, &num3) ;
            matx[num2].rdelay1 = num3 ;
            break ;
    case 19: fscanf(wq,"%d %d",&num2, &num3) ;
            matx[num2].fdelay1 = num3 ;
            break ;
    case 20: depth[0] = depth[0] + 1 ;
            break ;
    case 21: begint[0] = freft ;
            break ;
    case 22: (*(endt[0])).tptr = freft ;
            break ;
    case 23: endt[0] = freft ;
            break ;
    case 24: freft = (*freft).tptr ;
            break ;
    case 25: (*(endt[0])).tptr = NULL ;
            break ;
    case 26: fscanf(wq,"%d",&num2) ;
            (*(endt[0])).dptr = &(desc[num2]) ;
            break ;
    case 27: fscanf(wq,"%d",&num2) ;
            (*(endt[0])).newval = num2 ;
            break ;
    case 28: fscanf(wq,"%d %d",&num2, &num3) ;
           /*     fscanf(wq, "%c",&z) ;     if output names */
            z=getc(wq);
            z=getc(wq);
                  /* mess up                                  */
            sym[num2].name[num3] = z ;
            break ;
    case 29: fscanf(wq,"%d %d",&num2, &num3) ;
            sym[num2].index = num3 ;
            break ;
    case 30: sav_write = 0 ;
            break ;
    case 31: fscanf(wq,"%d",&num2) ;
            num_print = num2 ;
            break ;
    case 32: out_interval = 1 ;
            break ;
    case 33: fscanf(wq,"%d %d",&num2, &num3) ;
            desc[num2].pfunc = pnfn[num3] ;
            break ;
    case 50: endread = 1 ;
            break ;
    default: printf(" error in input data decoding\n") ;
   }
}     /* end while */
```

```
fclose(wp) ;
/*-------------------------------------------------------------*/


 wp = fopen(argv[2],"w") ; /*   is the output data file      */
                           /* connections                     */

 printf("        Welcome to MultiSimPC \n");
 printf(" \n");

 time = -1 ;
 timing_wheel = -1 ;
 sav_time = 0 ;

 printf(" Please enter Simulation time: ") ;
 scanf("%d",&sim_time) ;
 printf(" \n") ;

/*------------INITIALIZE INPUT DESCS. IN TIMING WHEEL------------*/
 for ( i = 0; i < num_input; i = i + 1 )
  {
   ptr = &(desc[i]) ;      /* beginning descs. = input descs.*/
   (*((*ptr).pfunc))(i) ;             /* call to read_input */
  }
/*-------------------------------------------------------------*/
 timing_wheel = 0 ;
/*-------------------------------------------------------------*/


/*--------------------BEGIN TIMING WHEEL-----------------------*/
 while ( time <= sim_time )
  {                                 /* begin while time < sim time */

   if (timing_wheel >= mdelay)
     timing_wheel = 0 ;             /* Timing wheel is circular    */

   while ( timing_wheel <  mdelay )
    {                              /* begin 1 loop of timing_wheel */
     time = time + 1 ;

     if (time > sim_time)
       break ;

     sav_depth = depth[timing_wheel] ;
     savt = begint[timing_wheel] ;        /* ptr to first element*/

     while ( depth[timing_wheel] != -1)
      {                /* while all row slots have been updated  */


       /*------------------BEGIN UPDATE---------------------------*/
       /* All descriptors connected to the current descriptor are*/
       /* updated with the 'future value' from the timing wheel  */
       /* if the present value differs from the 'future value'   */

       fwdt = begint[timing_wheel] ;
```

```
ptr = (*fwdt).dptr ;

beqint[timing_wheel] = (*(beqint[timing_wheel])).tptr ;
/* beqint points to the next element in the current row   */


/*-------------SCHEDULE INPUT READING---------------------*/
if (((*fwdt).sflag) == 1)
 {    /* if scheduling flag is 1, schedule the descriptor */

  (*((*ptr).pfunc))((*ptr).parent) ;

  (*fwdt).sflag = 2 ;
                              /* de-assert scheduling flag */
 }
/*-----------------------------------------------------*/


depth[timing_wheel] = depth[timing_wheel] - 1 ;

flag = 0 ;     /* flag = 1 indicates a multi-output desc. */

sav_value = (*fwdt).newval ;
          /* sav_value = future value from tim. wheel     */
          /* i.e. value to be replaced for present output */

while (1)
 {       /* begin while C-list for each output is updated */

  sav_ptr = ptr ;

  if (sav_value != -1)
   {

    if (((*ptr).present_output) != sav_value)
     {  /* change has occured */

      ptr = (*ptr).header ;

      if (ptr != sav_ptr)
       {   /* if not end of circular list, then begin     */

        (*ptr).param[(*sav_ptr).h_value] = sav_value ;
         /* input no. pointed to by the desc. is updated  */

        prev_ptr = ptr ;

        switch ((*sav_ptr).h_value)
         {
          case 0 : ptr = (*ptr).right0 ;
                   field_no = (*prev_ptr).r_value[0] ;
                   break ;
          case 1 : ptr = (*ptr).right1 ;
                   field_no = (*prev_ptr).r_value[1] ;
         }
```

```
            while (1)
             { /* while begin */

               if (ptr == sav_ptr)
               /* if end of circular list, then get out */
                 break ;

               (*ptr).param[field_no] = sav_value ;
               /* update input with 'future value' */

               prev_ptr = ptr ;

               switch (field_no)
                {
                 case 0 : ptr = (*ptr).right0 ;
                          fx = (*prev_ptr).r_value[0] ;
                          break ;
                 case 1 : ptr = (*ptr).right1 ;
                          fx = (*prev_ptr).r_value[1] ;
                }

               field_no = fx ;

             } /*      end while                          */
           }   /*      end then (if not end of C-list)    */
         }     /*      end if change has occured          */
       }       /*      if sav_value != -1                 */

      if (flag == 1)
       {
        if ((*nwptr).newptr == NULL)
          break ;          /* if nstack has no more extension */
        nwptr = (*nwptr).newptr ;
       }

      else
       {
        if ((*fwdt).nptr == NULL)
          break ;
        nwptr = (*fwdt).nptr ;
        flag = 1 ;
              /* flag is asserted for a multi-output function */
       }

      ptr = (*sav_ptr).ext_ptr ;      /* if multi-output case */

      sav_value = (*nwptr).newvalue ;

     }        /* end update of all descs. in C-list */
   }          /* end of update of one row of T.W.   */
  /*--------------END UPDATE-------------------------------*/

depth[timing_wheel] = sav_depth ;
begint[timing_wheel] = savt ;
```

```
/*-------------------EXECUTION PHASE-------------------------*/
/* Second pass -Schedule the descs. in C-list & insert in    */
/* proper time slot if output of current desc. has changed   */


while ( depth[timing_wheel] != -1)
 {          /* begin execution of one row of timing wheel   */


  fwdt = beginf[timing_wheel] ;
  beginf[timing_wheel] = (*(beginf[timing_wheel])).fptr ;

  ptr = (*fwdt).dptr ;

  depth[timing_wheel] = depth[timing_wheel] - 1 ;

  sav_value = (*fwdt).newval ;
              /* future value */

  flag = 0 ;        /* flag = 1 indicates multi-output desc. */


  while (1)
   {       /* begin while not end of C-list for all outputs */

    sav_ptr = ptr ;

    if (sav_value != -1)
     {

      if (((*ptr).present_output) != sav_value)
       {                                  /* change has occured */
        (*ptr).present_output = sav_value ;
                                    /* update present output */
        ptr = (*ptr).header ;

        if (ptr != sav_ptr)
         {          /* if not end of C-list, then begin      */


          /*---SCHEDULE DESC. ------------------------------*/
          parent_ptr = &(desc[(*ptr).parent]) ;
          (*((*parent_ptr).pfunc))(0,duminp,f_out) ;
          /*-----------------------------------------------*/

          fmode() ;      /* mode simulation */

          fdel_ins((*sav_ptr).h_value);
                   /* find delays and insert in proper slot */

          prev_ptr = ptr ;

          switch ((*sav_ptr).h_value)
           {
```

```
        case 0 : ptr = (*ptr).right0 ;
                 field_no = (*prev_ptr).r_value[0] ;
                 break ;
        case 1 : ptr = (*ptr).right1 ;
                 field_no = (*prev_ptr).r_value[1] ;
       }

      while (1)
       { /* while begin */

        if (ptr == sav_ptr) /* if end of C-list, quit   */
          break ;


        /*---SCHEDULE DESC. -get all parameters first --*/
        parent_ptr = &(desc[(*ptr).parent]) ;
        (*((*parent_ptr).pfunc))(0,duminp,f_out) ;
        /*-------------------------------------------*/

        fmode() ;       /* simulate mode behavior */

        fdel_ins(field_no) ;
              /* find delays and insert in proper slot */

        prev_ptr = ptr ;

        switch (field_no)
         {
          case 0 : ptr = (*ptr).right0 ;
                   fx = (*prev_ptr).r_value[0] ;
                   break ;
          case 1 : ptr = (*ptr).right1 ;
                   fx = (*prev_ptr).r_value[1] ;
          }

         field_no = fx ;

        } /*     end while                          */
      }   /*     end then  (if not end of C-list) */
    }     /* end if change has occured             */
  }       /*  end if sav_value != -1               */


if (flag == 1)       /* multi output case          */
 {
  if ((*nwptr).newptr == NULL)
   {
   /* fref storage for nstack */

    (*nwptr).newptr = fref ;
    fref = nwptr ;

    break ;  /* If all outputs have been handled, quit */
   }

  /* free storage for the previous nstack */
```

```
            pptr = nwptr ;
            nwptr = (*nwptr).newptr ;
            (*pptr).newptr = free ;
            free = nptr ;
           }

          else     /* if flag is 0 */
           {
            if ((*fwdt).nptr == NULL)
              break ;
            nwptr = (*fwdt).nptr ;
            flag = 1 ;      /* multi-output case   */
           }

          ptr = (*sav_ptr).ext_ptr ;
          sav_value = (*nwptr).newvalue ;

         }   /* end (C-list scan for all outputs) */


        /*-------------Free storage for current tstack ----------*/
        tempt = freet ;
        freet = fwdt ;
        (*freet).tptr = temot ;
        /*-----------------------------------------------------*/


       }     /* end of one row of timing wheel    */


      write_output() ;       /* print results */

      begint[timing_wheel] = NULL ;
      endt[timing_wheel] = NULL ;
      timing_wheel = timing_wheel + 1 ;
                             /* move to next row of timing wheel */

     }       /* end of i loop (vertical) of timing wheel */
   }         /* end time < sim time                      */

  fclose(rp) ;
  fclose(wp) ;

 }  /* end of main program */
/*-------------------------------------------------------------*/


/*-----------------fmode()-------------------------------------*/
/* It finds outputs for each descriptor according to its mode    */
fmode()
 {
  int i, mode ;
  struct descrpt *xtptr ;

  xtptr = parent_ptr ;
  for (i = 0; i <  num_outs ; i = i + 1)
```

```
    {
     mode = (*xtptr).param[6] ;
     switch (mode)
      {
       case 0 : f_out[i] = f_out[i] ;
                break;
       case 1 : if (f_out[i] == 0)
                  f_out[i] = 2 ;
                break;
       case 2 : if (f_out[i] == 1)
                  f_out[i] = 2 ;
                break;
       case 3 : f_out[i] = 0 ;
                break ;
       case 4 : f_out[i] = 1 ;
      }
     xtptr = (*xtptr).ext_ptr ;
    }
 }
/*-------------------------------------------------------------*/


/*--------------------FDEL_INS() -------------------------------*/
/* This procedures finds the delays applicable to a function     */
/* connected to the output of the current descriptor. The input  */
/* number to which the current decriptor is connected is supplied*/

fdel_ins(inp)

int inp ;         /* inp = input number to which the current desc.*/
 {                /* connected  */

  int i, j, k ;
  struct matrix *mtptr ;


/*----------------------COMPUTE DELAYS-----------------------------*/
  if (f_out[0] == 1)
   {
    if (inp == 0)
      del[0] = (*ptr).param[2] ;          /* del-0 = rise del(0,0) */
    else
      del[0] = (*ptr).param[4] ;          /* del-0 = rise del(1,0) */
   }
  else    /* if first output is 1 */
   {
    if (inp == 0)
     del[0] = (*ptr).param[3] ;           /* del-0 = fall del(0,0) */
    else
     del[0] = (*ptr).param[5] ;           /* del-0 = fall del(1,0) */
   }


  if ((*ptr).mptr != NULL)               /* multi-output case     */
   {
    mtptr = (*ptr).mptr ;
```

```
    for (i = 1; i < num_outs; i = i + 1)
     {
      if (f_out[i] == 1)
       {
        if (inp == 0)
          del[i] = (*mtptr).rdelay0 ;    /* del-i = rise del(0,i) */
        else
          del[i] = (*mtptr).rdelay1 ;    /* del-i = rise del(1,i) */
       }
      else    /* if ith output is 0 */
       {
        if (inp == 0)
          del[i] = (*mtptr).fdelay0 ;    /* del-i = rise del(0,i) */
        else
          del[i] = (*mtptr).fdelay1 ;    /* del-i = fall del(1,i) */
       }
      if ((*mtptr).matptr == NULL)
        break ;
      else
        mtptr = (*mtptr).matptr ;
     }
   }
/*-----------------END COMPUTE DELAYS---------------------------*/


/* For a multi-output function, different delays are possible.   */
/* Insert in proper slot for each different delay                */

  for (i = 0; i < num_outs; i = i + 1)
   {
    if (del[i] != -1)   /* delay = -1 means input and output are */
     {                  /* not related.                          */

      delay = del[i] ;
      ff_out[i] = f_out[i] ;


      /*--------------------Check for identical delays -----------*/
      for (j = i+1; j < num_outs; j = j + 1)
       {
        if (del[j] == delay)       /* For identical delays on     */
          ff_out[j] = f_out[j] ;   /* different outputs, function */
        else                       /* should be inserted only once*/

          ff_out[j] = -1 ;         /* -1 output means, keep the   */
       }                           /* output                      */
       /*-------------------------------------------------------*/


      /*-------Eliminate the delay cases which have been covered */
      for (k = 0; k < num_outs; k = k + 1)
       {
        if (del[k] == delay)
          del[k] = -1 ;
       }
       /*-------------------------------------------------------*/
```

```
      insert() ;        /* insert in proper slot */
     }

    else                /* if delay = -1 */
      ff_out[i] = -1 ;
   }
 }
/*-------------------------------------------------------------*/


/*---------------------INSERT()--------------------------------*/
/* This procedure inserts the function in proper slot in T.W.    */

insert()
 {
  int i,j,k, slot_no ;

  slot_no = (delay + timing_wheel)%mdelay ;
  depth[slot_no] = depth[slot_no] + 1 ;

  if (endt[slot_no] != NULL)
    (*(endt[slot_no])).tptr = freft ;
  else
    begint[slot_no] = freft ;

/* allocate storage for tstack */
  endt[slot_no] = freft ;
  freft = (*freft).tptr ;
  (*(endt[slot_no])).tptr = NULL ;
/*-----------------------------*/


  (*(endt[slot_no])).dptr = parent_ptr ;
  (*(endt[slot_no])).newval = ff_out[0] ;

  if (num_outs > 1)       /* multi-output cases */
   {
    (*(endt[slot_no])).nptr = fref ;
    /* allocate storage for newstack */

    (*fref).newvalue = ff_out[1] ;

    nwlpt = fref ;

    fref = (*fref).newptr ;

    if (num_outs > 2)
     {
      k = 0 ;
      j = 2 ;
      while (k == 0)
       {
        (*nwlpt).newptr = fref ;
        nwlpt = fref ;
```

```
        (*fref).newvalue = ff_out[j] ;

        fref = (*fref).newptr ;
        j = j + 1 ;
        if (j == num_outs)
          break ;
       }
     }
    (*nwipt).newptr = NULL ;
   }

  else       /* if single output function */

   (*(endt[slot_no])).ptr = NULL ;
            /* put new value in the slot */
 }
/*-------------------------------------------------------------*/


/*-------------------------READ INPUT routine--------------------*/
read_input(ddnum)
int ddnum ;
{
 int i, order, k, slotn, p0, p2 ;

/*  determine inout order i.e. which input is to be read  */

 p0 = desc[ddnum].param[0] ;
 p2 = desc[ddnum].param[2] ;
 for ( i = 0; i < num_input ; i = i + 1 )
  { if ( p0 == inptc[i]  )
      break ;
   }
 order = i ;
                  /* inout order found so quit searching */
 if (sav_time != time )
  {
   sav_time = time ;
   for ( i = 0; i < num_input ; i = i + 1 )
    {
     fscanf(rp,"%d",&k)  ;
     inpt[i] = k ;
    }
  }
 f_out[0] = inpt[order] ;
 slotn = (timing_wheel + p2)%(mdelay) ;
 depth[slotn] = depth[slotn] + 1 ;

 if (endt[slotn] != NULL )
   (*(endt[slotn])).ptr = freft ;
 else
   begint[slotn] = freft ;

/* allocate storage for tstack */
 endt[slotn] = freft ;
 freft = (*freft).ptr ;
```

```
 (*(endt[slotn])).totr = NULL ;

/*-----------------------------*/
 (*(endt[slotn])).dotr = &(desc[order]) ;
 (*(endt[slotn])).newval = f_out[0] ;
 (*(endt[slotn])).notr = NULL ;
 (*(endt[slotn])).sflag = 1 ;
}
/*-------------------------------------------------------------*/



/*---------------------WRITE OUTPUT routine---------------------*/
write_output ()

/* num_print  contains the number of lines to be printed        */
/* out_interval is interval after which value is to be printed  */
/* sym[].index contains indices of descs. representing output   */

{
 int i ;
 if (sav_write == 0)
  {
   fprintf(wp," time");
   for (i = 0; i < num_print ; i = i + 1)
     fprintf(wp," %5s",sym[i].name) ;
   fprintf(wp,"\n") ;
   fflush(wp) ;
  }
 sav_write = 1 ;
   fprintf(wp, " %3d ",time) ;

 for (i = 0; i < num_print ; i = i + 1)
   fprintf(wp," %5d",desc[(sym[i].index)].present_output) ;
 fprintf(wp,"\n") ;
 fflush(wp) ;
}
/*-------------------------------------------------------------*/

#include "block"

/*-----------------------INVERTER--------------------------------*/
INVERT(flg,  inpx, ou)
int flg , *inpx, *ou ;
 {
  if (flg == 0)
    indata(inpx, 1) ;

  switch((*inpx))
  {
   case 0 : (*ou) = 1 ;
            break ;
   case 1 : (*ou) = 0 ;
            break ;
   case 2 : (*ou) = 2 ;
  }
```

```
 num_outs = 1 ;
 }
/*-------------------------------------------------------------*/


/*-----------------------AND()---------------------------------*/
 AND(flg , inpx, ou)
 int flg , *inpx, *ou ;
 {
  int s ;

  if (flg == 0)
    indata(inpx, 2) ;

  s = (*inpx) + (*(inpx + 1)) ;
  switch(s)
   {
    case 0 : (*ou) = 0 ;
             break ;
    case 1 : (*ou) = 0 ;
             break ;
    case 2 : if ((*inpx) == 1)
               (*ou) = 1 ;
             else
              (*ou) = 0 ;
             break;
     default : (*ou) = 2 ;
    }
   num_outs = 1 ;
  }
/*-------------------------------------------------------------*/


/*-----------------------OR()----------------------------------*/
 OR(flg ,inpx, ou)
 int flg , *inpx, *ou ;
 {
  int s ;

  if (flg == 0)
    indata(inpx, 2 ) ;

  s = (*inpx) + (*(inpx+1)) ;
  switch(s)
   {
    case 0 : (*ou) = 0 ;      /* ou is pointer to f_out */
             break ;
    case 1 : (*ou) = 1 ;
             break ;
    case 2 : if ((*inpx) == 1)
               (*ou) = 1 ;
             else
              (*ou) = 2 ;
             break;
     case 3 : (*ou) = 1 ;
             break ;
```

```
      default : (*ou) = 2 ;
     }
    num_outs = 1 ;
   }
/*-------------------------------------------------------------*/


/*--------------------indata()---------------------------------*/
/* input data retrieval for a function                        */
 indata(inpx,inns)
 int *inpx, inns ;
 /* inox = input data array,   inns = number of input data     */
  {
   struct descrpt *tmptr ;
   int i ;

   if (inns >= 1)
    {
     (*inpx) = (*parent_ptr).param[0] ;
     inns = inns - 1 ;
     if (inns != 0)
      {
       (*(inpx+1)) = (*parent_ptr).param[1] ;
       inns = inns - 1 ;
      }
     i = 2 ;
     tmptr = parent_ptr ;
     while (inns != 0)
      {
       tmptr = (*tmptr).ext_ptr ;
       (*(inpx + i))= (*tmptr).param[0] ;
       inns = inns - 1;
       i = i + 1 ;
       if (inns != 0)
        {
         (*(inpx + i)) = (*tmptr).param[1] ;
         i = i + 1 ;
         inns = inns - 1 ;
        }
      }     /* end while */
    }
  }
/*-------------------------------------------------------------*/


/*-------------------------------------------------------------*/
hashf(s)    /* forms hash value for string s */
char *s;
{
  int hashval ;

  for (hashval = 0;  *s != '\0' ; )
     hashval += *s++ ;              /* hash ID name into an index */

  test(&hashval);                   /* and test for collisions    */
```

```
  hashtable[hashcount]=hashval;
  hashcount++;

  return (hashval) ;
}
/*-----------------------------------------------------------------*/


/*----------------------------test()-------------------------------*/
test(value)
int *value;
   {
   int i;
   for (i=0; i<hashcount; i++)
      {
      if (hashtable[i]==(*value))         /* collision?    */
         {
         (*value)=(*value)+11;            /* yes, add a prime number */
         test(value);                     /* and test this one */
         }
      }
   }
/*-----------------------------------------------------------------*/


/*--------------------------getname()------------------------------*/
getname()   /* returns the name in tokenbuff */
{
 int i, c , flag, delimiter ;
 i = 0 ;
 delimiter = -1 ;
 flag = 0 ;
 while(( delimiter < 1) || (flag == 0))
  {
   c = fgetc(rp) ;
   switch (c)
    {
     case ' ' : delimiter = 1 ;
                break ;
     case ',' : delimiter = 1 ;
                break ;
     case '\n': delimiter = 1 ;
                endinoutname = 1 ;
                flag = 1 ;
                break ;
     default : flag = 1 ;
               delimiter = 0 ;
    }
   if (delimiter == 0)
    {
      if (i <= 6)
       {
        tokenbuff[i] = c ;
        i = i + 1 ;
       }
```

```
    }
  }
  tokenbuff[i] = '\0' ;
}
/*-----------------------------------------------------------------*/
```

APPENDIX E.

SIMULATOR AUXILIARY FILES

FTYPE — Primitive function type declarations. An **#include** file.

FADDR — Primitive function pointer initializations. An **#include** file.

BLOCK — The C-language behavioral descriptions for the primitives. An **#include** file.

```
int NAND();
int NOR();
int ANDTHRE();
int ORTHREE();
int NANDTHR();
int SRBLOCK();
int RETDBLO();
int ANDFOUR();
int ORFOUR();
int EXOR();
```

The Auxiliary File **FTYPE**

```
pnfn[3] = NAND;
pnfn[4] = NOR;
pnfn[6] = EXOR;
pnfn[7] = ANDTHRE;
pnfn[8] = NANDTHR;
pnfn[9] = SRBLOCK;
pnfn[10]= RETDBLO;
pnfn[11]= ANDFOUR;
pnfn[12]= NANDFOU;
pnfn[13]= ORTHREE;
pnfn[14]= ORFOUR;
pncnt = 15;
```

The Auxiliary File **FADDR**

```
/*************************************************************
 *         block level primitive description file            *
 *************************************************************/


/*--------------------------NAND GATE-----------------------------------*/
NAND(flg ,inpx,ou)
int flg , *inpx, *ou ;
{
 int i ;

 if (flg == 0)
   indata ( inpx, 2) ;

 i = (*inpx) + (*(inpx + 1)) ;
 switch(i)
  {
   case 0 : (*ou) = 1 ;
            break ;
   case 1 : (*ou) = 1 ;
            break ;
   case 2 : if ((*inpx) == 1)
              (*ou) = 0 ;
            else
              (*ou) = 1 ;
            break ;
    default : (*ou) = 2 ;
  }
 num_outs = 1 ;
}
/*-------------------------------------------------------------------*/


/*--------------------------NOR  GATE-----------------------------------*/
NOR(flg ,inpx, ou)
int flg , *inpx, *ou ;
{
 int i ;

 if (flg == 0)
   indata ( inpx, 2) ;

 i = (*inpx) + (*(inpx + 1)) ;
 switch(i)
  {
   case 0 : (*ou) = 1 ;
            break ;
   case 1 : (*ou) = 0 ;
            break ;
   case 2 : if ((*inpx) == 1)
              (*ou) = 0 ;
            else
              (*ou) = 2 ;
            break ;
    default : if (((*inpx) == 1) || ((*(inpx + 1)) == 1))
                (*ou) = 0;
```

```
                else
                  (*ou) = 2 ;
  }
 num_outs = 1 ;
}
/*-------------------------------------------------------------*/


/*---------------------THREE INPUT AND GATE--------------------*/
ANDTHRE(flg ,inpx, ou)
int flg , *inpx, *ou ;
 {
  int inn[2] ;
  if (flg == 0)
   indata(inpx, 3) ;

  AND(1,inpx, inn) ;
  inn[1] = (*(inpx + 2)) ;
  AND(1,inn, ou) ;

  num_outs = 1 ;
 }
/*-------------------------------------------------------------*/


/*---------------------THREE INPUT NAND GATE-------------------*/
NANDTHR(flg ,inpx, ou)
int flg , *inpx, *ou ;
 {
  int m ;

  if (flg == 0)
    indata(inpx, 3) ;

  ANDTHRE(1,inpx, &m) ;
  INVERT(1,&m, ou) ;

  num_outs = 1 ;
 }
/*-------------------------------------------------------------*/


/*---------------------SRBLOCK---------------------------------*/
SRBLOCK(flg ,inpx, ou)
int flg , *inpx, *ou ;
 {
  int inn[2] ;

  if (flg == 0)
    indata(inpx, 3) ;

  NAND(1, inpx, inn) ;
  inn[1] = (*(inpx + 2)) ;
  (*ou) = (*inn) ;
  NAND(1, inn, (ou+1)) ;
  (*(ou+2)) = (*(ou + 1)) ;      /* Y output is same as Q'  */
```

```
  num_outs = 3 ;
 }
/*-------------------------------------------------------------*/


/*--------------------------RETDBLO()--------------------------*/
RETDBLO(flg ,inpx, ou)
int flg , *inpx, *ou ;
 {
  int i, inn[3] ;

  if (flg  == 0)
    indata(inpx, 6) ;

  NAND(1,(inpx + 3),inn) ;        /* A =  NAND(X1, X2) */
                                  /* inn[0] = A        */
  inn[1] = (*inpx) ;              /* inn[1] = clr      */
  inn[2] = (*(inpx + 2)) ;        /* inn[2] = CLK      */
  NANDTHR(1,inn, (ou+3));         /* X2 = NANDTHRE(A,clr,CLK):  */
  NANDTHR(1,(inpx + 2),&(inn[2]));/* B = NANDTHRE(X2, CLK, X1)*/
                                  /* inn(2] = B                 */
  NAND(1,(inpx + 4),ou) ;         /* Q = NAND(X2, X3) ;         */
  inn[0] = (*ou) ;                /* inn[0] = Q                 */
  inn[1] = (*inpx) ;              /* inn(1] = clr               */
  NANDTHR(1, inn,(ou+1));         /* Qc = NANDTHRE(Q,clr,B);    */
  inn[0] = (*(inpx + 1)) ;        /* inn[0] = D                 */
  NANDTHR(1, inn,(ou+2));         /* X1 = NANDTHRE(B, clr, D)   */
  (*(ou+4)) = (*(ou + 1)) ;       /* X3 = Qc                    */
  num_outs = 5 ;
 }
/*-------------------------------------------------------------*/


/*--------------------FOUR  INPUT AND GATE---------------------*/
ANDFOUR(flg ,inpx, ou)
int flg , *inpx, *ou ;
 {
  int inn[2];

  if (flg == 0)
    indata(inpx, 4) ;

  ANDTHRE(1, inpx, inn) ;
  inn[1] = (*(inpx + 3)) ;
  AND(1, inn, ou) ;

  num_outs = 1 ;
 }
/*-------------------------------------------------------------*/


/*-----------------------NANDFOUR()----------------------------*/
NANDFOU(flg ,inpx, ou)
int flg , *inpx, *ou ;
 {
```

```
  int m ;

  if (flg == 0)
    indata(inpx, 4) ;

  ANDFOUR(1,inpx, &m) ;
  INVERT(1, &m, ou) ;

  num_outs = 1 ;
 }
/*-------------------------------------------------------------*/


/*-------------------------ORTHREE()---------------------------*/
ORTHREE(flg ,inpx, ou)
int flg , *inpx, *ou ;
 {
  int inn[2];

  if (flg == 0)
    indata(inpx, 3) ;

  OR(1,inpx, inn) ;
  inn[1] = (*(inpx + 2)) ;
  OR(1, inn, ou) ;

  num_outs = 1 ;
 }
/*-------------------------------------------------------------*/



/*--------------------FOUR INPUT OR GATE-----------------------*/
ORFOUR(flg ,inpx, ou)
int flg , *inpx, *ou ;
 {
  int inn[2] ;

  if (flg == 0)
    indata(inpx, 4) ;

  ORTHREE(1,inpx, inn) ;
  inn[1] = (*(inpx + 3)) ;
  OR(1, inn, ou) ;

  num_outs = 1 ;
 }
/*-------------------------------------------------------------*/


/*-------------------------EXOR GATE---------------------------*/
EXOR(flg ,inpx, ou)
int flg , *inpx, *ou ;
{
 int i ;
```

```
 if (flg == 0)
   indata(inpx, 2) ;

 i = (*inpx) + (*(inpx + 1)) ;
 switch(i)
  {
   case 0 : (*ou) = 0 ;
            break ;
   case 1 : (*ou) = 1 ;
            break ;
   case 2 : if ((*inpx) == 1)
              (*ou) = 0 ;
            else
              (*ou) = 2 ;
            break ;
    default : (*ou) = 2 ;
  }
 num_outs = 1 ;
}
/*-----------------------------------------------------------------*/
```

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Dr. H. B. Rigas Chair, Electrical and Computer Engineering Department (Code 62) Naval Postgraduate School Monterey, California 93943	5
4.	Robert Wasilausky (Code RW24) Naval Oceanographic Systems Command 271 Catalina Boulevard San Diego, California 92512	1
5.	LT J. Scott Kelly USN 2914 State Hill Road Apartment D8 Wyomissing, Pennsylvania 19610	1
6.	LCDR Julio Albuquerque, Brazilian Navy SMC 2273 Naval Postgraduate School Monterey, California 93943	1

17898 2

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-50

220293

Thesis
K2853 Kelly
c.1 MultiSimPC: a multi-
level logic simulator
for microcomputers.

220293

Thesis
K2853 Kelly
c.1 MultiSimPC: a multi-
level logic simulator
for microcomputers.

MultiSimPC:
3 2768 000 70260 9
DUDLEY KNOX LIBRARY